Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

1º trabalho laboratorial

Authors:

Elgner Ramos up201208090
João Fidalgo up201605237
Ricardo Boia up201505244

November 4, 2018

Contents

1 Sumário

Desenvolvido no âmbito da unidade curricular de Redes de Computadores, este trabalho teve como objetivo a criação de uma aplicação capaz de transferir ficheiros entre computadores através de uma porta série.

Pensamos que os objetivos do trabalho foram cumpridos, visto que a transferência é feita sem perdas e com as caraterísticas estabelecidas no guião.

2 Introdução

Este projeto foi desenvolvido com o objetivo de implementar um protocolo de ligação de dados fiável entre dois sistemas ligados por um cabo série, e testá-lo através de uma aplicação simples de transferência de ficheiros. Já este relatório foi elaborado para explicar em linguagem corrente as funcionalidades do programa, recorrendo a esta estrutura:

- Arquitetura - explicação dos blocos funcionais e interfaces presentes.
- Estrutura do código - API, principais estruturas e funções, e a sua relação com a arquitetura.
- Casos de uso principais - identificação e exibição das sequências de chamada das funções.
- Protocolo de ligação lógica - identificação dos principais aspectos funcionais e descrição da estratégia de implementação destes aspectos.
- Protocolo da aplicação - identificação dos principais aspectos funcionais e descrição da estratégia de implementação destes aspectos.
- Validação - descrição dos testes efectuados com apresentação quantificada dos resultados.
- Eficiência do protocolo de ligação de dados - caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.
- Conclusões - síntese da informação apresentada nas secções anteriores e reflexão sobre os objectivos de aprendizagem alcançados.

3 Arquitetura

O programa está dividido em dois blocos funcionais:

- Ligação de dados (*linkLayer*) - onde se encontram as funções diretamente responsáveis, não só pelo estabelimento e terminação da ligação, mas também pelo cumprimento das regras do protocolo durante a transmissão de tramas.
- Aplicação (*appLayer*) - utiliza as funções existentes na *linkLayer* para transmissão de um ficheiro entre dois sistemas.

4 Estrutura do código

4.1 Ligação de dados (*linkLayer*)

4.1.1 Principais funções

As funções *llopen* e *llclose* foram divididas em duas sub-funções, uma para o emissor e outra para o recetor.

- llopen - inicia a ligação.
- llwrite - constrói a trama, faz o seu *stuffing* e envia-a.
- llread - recebe a trama, faz o *destuffing* e valida-a.
- llclose - termina a ligação.

4.1.2 Principal estrutura

Apenas existe uma estrutura na camada de ligação de dados mas é a mais importante do programa. Nesta estrutura são guardadas as variáveis necessárias para a transmissão do ficheiro - descritor de ficheiro, *port*, *baudrate*, número de sequência... - e as estatísticas - número de RRs e REJS enviados/recebidos e o tempo decorrido durante a transmissão.

```
typedef struct
{
    int fd;                       /*Descritor de ficheiro */
    char *port;                   /*Dispositivo /dev/ttySx, x = 0, 1*/
    int baudRate;                 /*Velocidade de transmissão*/
    unsigned int sequenceNumber;  /*Número de sequência da trama: 0, 1*/
    unsigned int timeout;         /*Valor do temporizador:*/
    unsigned int numTransmissions; /*Número de tentativas em caso de falha*/
    unsigned char *frame;         /*Trama */

    char *fileName; /*nome do ficheiro */
    int fileSize; /*tamanho do ficheiro */

    unsigned int nRR; /*número de RRs enviados/recebidos */
    unsigned int nREJ; /*número de REJs enviados/recebidos */
    double totalTime; /*tempo decorrido durante a transmissão do ficheiro */
} LinkLayer;
```

Figure 1: Estrutura *LinkLayer*

4.2 Aplicação (*appLayer*)

4.2.1 Principais funções

- send, sendControl e sendData - preparam os pacotes de controlo e de dados a enviar e passam-nos para o *llwrite*.
- receive - recebe e processa os pacotes recebidos por *llread*.

4.2.2 Principais estruturas

São três as estruturas declaradas na camada da aplicação:

- ApplicationLayer - indica se o computador é o emissor ou o recetor.

- TLV - parâmetro utilizado nos pacotes de controlo.
- ControlPacket - guarda a informação sobre um pacote de controlo - campo de controlo e parâmetros.

```c
typedef struct
{
    Status status; /*TRANSMITTER ou RECEIVER */
} ApplicationLayer;

typedef struct
{
    unsigned int type; /* 0 - tamanho do ficheiro, 1 - nome do ficheiro */
    unsigned char lenght; /* tamanho do campo value */
    char *value; /* valor do parâmetro */
} TLV;

typedef struct
{
    unsigned int controlField; /* 2 - start, 3 - end */
    TLV *parameters; /* estrutura declarada em cima */
} ControlPacket;
```

Figure 2: Estruturas declaradas em *appLayer*

4.2.3 Macros principais

- BYTESTOSEND - número de bytes enviados de cada vez.

4.3 Utilities

4.3.1 Funções principais

Este ficheiro contém funções que são utilizadas pelas duas camadas principais. Destas funções destacam-se:

- validateCommand - verifica se a estrututa das tramas de comando e das tramas *UA* está correta.
- validateFrame - verifica se a estrututa da trama de informação recebida está correta.
- receiveResponse - recebe uma resposta do recetor - *RR0, RR1, REJ0 ou REJ1* - e verifica a sua estrutura.
- stuffing e destruffing - realiza o *stuffing* e o *destuffing* das tramas, respetivamente.
- isValidBcc2 - verifica se o BCC2 das tramas de informação é o esperado.

4.3.2 Variáveis globais

- SETUP, UA, DISC, RR0, RR1, REJ0, REJ1 - *const arrays* que representam as tramas de supervisão e não numeradas a enviar.

4.3.3 Macros principais

- ERRORPROBABILITY1 - probabilidade de simular um erro na leitura do BCC1 das tramas de informação recebidas.
- ERRORPROBABILITY2 - probabilidade de simular um erro na leitura do BCC2 das tramas de informação recebidas.

5 Casos de uso principais

Na execução do programa é necessário o utilizador colocar a porta série a ser utilizada (ex: **/dev/ttyS0**), o *status* (**0** para o emissor e **1** para o recetor) e o nome do ficheiro (ex: **pinguim.gif**). De seguida, é dada a escolher o *baudrate* a que se vai realizar a transmissão do ficheiro. A transmissão realiza-se com a seguinte sequência de eventos:

- O emissor e o rector abrem a porta série e configuram a estrutura *termios*,
- A ligação entre os dois é estabelecida (*llopen*),
- Os dados do ficheiro são enviadas pelo emissor.
- O recetor recebe os dados e guarda-os num ficheiro com o mesmo nome do ficheiro recebido.
- A ligação é terminada (*llclose*).

6 Protocolo de ligação lógica

6.1 Abertura da porta série

A abertura da porta de série é efetuada no início do programa na função **openPort**, utilizando a variável *port* guardada na estrutura de dados **LinkLayer** recebida como argumento. O descritor de ficheiro retornado pela função *open* é depois guardado na estrutura para ser utilizado para operações de *write* e *read*.

```
int openPort(LinkLayer *linkLayer);
```

6.2 Estabelecimento da ligação

O estabelecimento da ligação está atribuído às funções **llopen**, que optamos por dividir em duas: **llopenR**, que estabelece a ligação no recetor, e **llopenT**, que estabelece a ligação no emissor. A função **llopenR** recebe a mensagem de *SETUP* por parte do emissor e envia *UA* como resposta. Já a função **llopenT** envia a mensagem de *SETUP* e fica à espera da resposta do recetor durante um tempo predefinido (*timeout*). Se isto não acontecer, este processo pode repetir-se um número predefinido de vezes, que se for ultrapassado termina o programa (tanto o valor do *timeout* como o de número de tentativas estão guardados na estrutura **LinkLayer**). A validação das tramas *SET* e *UA* é feita pela função **validateCommand**.

```
int llopenT(LinkLayer *linkLayer);
int llopenR(LinkLayer *linkLayer);
```

6.3 *Byte stuffing/destuffing*

Ambas as funções (***stuffing*** e ***destuffing***) encontram-se no ficheiro utilities. O *stuffing* é feito imediatamente antes do envio dos dados, percorrendo o campo de informação e substituindo todos os octetos que forem iguais ao padrão de uma *flag* ou ao octeto de escape para um valor especificado no guião.

Já o *destuffing* é executado depois de serem recebidos os dados para repor os valores originais do campo de informação.

```
int stuffing(unsigned char* frame,int size);
int destuffing(unsigned char* frame,int size);
```

6.4 Envio de dados

O envio de dados é executado pela função **llwrite** que recebe o campo de informação como argumento (***buffer***), constrói a trama de informação, realiza o stuffing desta, envia-a para o recetor e fica à espera de uma resposta através da função **receiveResponse** – máquina de estados para tramas *REJ* e *RR*. . Se a reposta for *REJ* ou se esta não for recebida no tempo definido, a trama volta a ser enviada. Caso a resposta seja *RR*, então a trama foi enviada com sucesso e o programa prossegue para a próxima trama.

```
int llwrite(LinkLayer *linkLayer, unsigned char *buffer, int lenght);
```

6.5 Receção dos dados

A receção dos dados é efetuada pela função **llread** que recebe a trama com recurso a **validateFrame** – máquina de estados para tramas de informação. De seguida, realiza o seu *destuffing* e verifica-a. Nesta verificação é confirmado se o número de sequência é o esperado e se o *BCC2* está correto através da função *isValidBcc2*. Caso algum destes testes falhe é enviado um *REJ* para o emissor, caso contrário é enviado *RR*.

```
int llread(LinkLayer *linkLayer);
```

6.6 Terminação da ligação

A terminação da ligação é efetuada pela função **llclose** que tal como a **llopen** está dividida em **llcloseT** para o transmissor, e **llcloseR**, para o recetor. A função **llcloseT** envia uma mensagem de *DISC*, aguarda pela resposta do recetor, que também deverá ser um *DISC*, e finalmente envia um *UA*, terminando. Já a função **llcloseR** recebe uma mensagem de *DISC* por parte do emissor, envia como resposta um *DISC* e espera um *UA* como resposta, terminando quando a receber. O envio de todas esta tramas é controlado através do mesmo sistema de *timeout* utilizado no **llopen** e no **llwrite** e, tal como no **llopen**, a validação das tramas é feita pela função **validateCommand**.

```
int llcloseT(LinkLayer *linkLayer);
int llcloseR(LinkLayer *linkLayer);
```

7 Protocolo de aplicação

7.1 Envio de pacotes

O envio de pacotes é realizado pela função **send** e duas funções auxiliares, **sendControl** e **sendData**. Primeiro, a função **send** preenche a estrutra **ControlPacket** com as informações do pacote de controlo *start* - campo de controlo a dois e dois parâmetros (um para o tamanho do ficheiro e outro para o nome). Este estrutura é depois enviada para a função **sendControl** que irá construir um pacote de controlo com as informações lá contidas e enviá-la para o **llwrite**. De seguida, irão ser enviados os dados do ficheiro, **BYTESTOSEND** bytes de cada vez, com o auxílio de *sendData*, que tal como **sendControl**, irá contruir um pacote, desta vez de dados, e enviá-lo para **llwrite**. Finalmente é enviado o pacote de controlo *end* que irá seguir um processo idêntico ao de *start*, apenas alterando o campo de controlo de dois para três.

```
void send(LinkLayer *linkLayer);
int sendControl(LinkLayer *linkLayer, ControlPacket *controlPacket, int nParameters);
int sendData(LinkLayer *linkLayer, char *buffer, int size, int sequenceNumber);
```

7.2 Receção de pacotes

A receção de pacotes está ao cargo de **receive**. Primeiramente, esta função vai entrar num ciclo *do while*, onde **llread** é invocado, e apenas sairá quando o campo de controlo do pacote recebido (e guardado na estrutura **linkLayer**) for igual a dois, sinalizando que se trata do pacote de controlo *start*. De seguida, irá ler deste pacote o nome e o tamanho do ficheiro a receber e abrir o ficheiro para onde serão enviados os dados recebidos - com o mesmo nome do ficheiro recebido. Finalmente, a função entra noutro ciclo onde **llread** é novamente chamado, e donde irá sair apenas quando receber o pacote de controlo *end*. Enquanto isto não acontecer, os pacotes recebidos serão de dados que serão escritos no ficheiro anteriormente aberto. No final do ciclo, o ficheiro é fechado e a função termina.

```
void receive(LinkLayer *linkLayer);
```

8 Validação

Foram realizados três testes para validar o bom funcionamento do programa:

- Transmissão do ficheiro sem interrupções.
- Transmissão do ficheiro com interrupções.
- Transmissão do ficheiro com a introdução do ruído, através da utilização de um fio.

No primeiro e no terceiro, os resultados foram os esperados. Em ambos os casos, o ficheiro foi enviado por pacotes de 256 bytes e foram recebidos 45 *RRs*, sendo que no primeiro teste o número de *REJs* foi zero, e no terceiro este número era variável mas sempre maior que zero.

Em relação ao segundo teste, os resultado nem sempre foram os esperados, visto que por vezes o emissor terminava devido ao número máximo de tentativas ter sido ultrapassado, mesmo quando isso não era suposto acontecer.

9 Eficiência do protocolo de ligação de dados

Após terminado o programa, foram realizados testes de modo a caraterizar estatisticamente a eficiência (**S**) do protocolo implementado. Para cada conjunto de valores foram realizados dois testes, sendo a sua média utilizada para o desenho do gráfico. Os testes realizados foram os seguintes:

9.1 Variação da capacidade de ligação (*baudrate*)

Analisando o gráfico, é possível observar que a eficiência é inversamente proporcional à capacidade de ligação.

Figure 3: Gráfico de eficiência, variando a capacidade de ligação

9.2 Variação do tamanho da trama

Analisando o gráfico, é possível observar que a eficiência é diretamente proporcional ao tamanho da trama.

Figure 4: Gráfico de eficiência, variando o tamanho da trama

9.3 Variação do FER

Analisando o gráfico, é possível observar que quanto maior for a probabilidade de acontecer erros nos campos de proteção das tramas (**BCC1** e **BCC2**), menor é a eficiência.

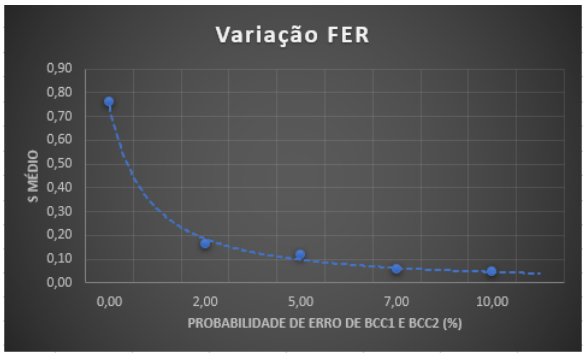

Figure 5: Gráfico de eficiência, variando o FER

9.4 Caraterização teórica do protocolo *Stop & Wait*

O protocolo utilizado foi o Stop Wait, um método de transmissão entre duas máquinas conectadas que garante que, para além da informação não ser perdida, esta é recebida na ordem correta. Isto acontece, pois cada vez que um pacote é enviado, o recetor tem de enviar uma resposta para confirmar que tudo correu como previsto para o emissor poder enviar o próximo pacote. Um grande problema deste protocolo é que quando a mensagem de confirmação enviada pelo recetor tiver erros, isto pode fazer com que o emissor assuma que ocorreram erros no envio, o que irá provocar o reenvio do pacote e, consequentemente, um problema de redundância no recetor.

10 Conclusões

Pensamos que os objetivos para este trabalho foram atingidos, visto que a transferência de dados é feita sem perda de informação. Com o desenvolvimento do projeto aprendemos conceitos que podem vir a ser úteis no futuro, como a transmissão de dados por porta série e a separação do programa em camadas. Em relação às maiores dificuldades encontradas, foi um desafio não só implementar o *timeout* no envio de tramas, como também entender a componente teórica do protocolo a ser utilizado para a transmissão dos dados

11 Anexo I

11.1 appLayer.h

```
1
2 #ifndef APPLAYER_H
3 #define APPLAYER_H
4
5 #include "linkLayer.h"
6 #include <unistd.h>
7 #include <stdio.h>
8 #include <string.h>
9
10 #define BYTESTOSEND 256
11
12 typedef enum
13 {
14   TRANSMITTER,
15   RECEIVER
16 } Status;
17
18 typedef struct
19 {
20   Status status; /*TRANSMITTER ou RECEIVER */
21 } ApplicationLayer;
22
23 typedef struct
24 {
25   unsigned int type; /* 0 - tamanho do ficheiro, 1 - nome do ficheiro */
26   unsigned char lenght; /* tamanho do campo value */
27   char *value; /* valor do parametro */
28 } TLV;
29
30 typedef struct
31 {
32   unsigned int controlField; /* 2 - start, 3 - end */
33   TLV *parameters; /* estrutura declarada em cima */
34 } ControlPacket;
35
36 void startAppLayer(LinkLayer *linkLayer, ApplicationLayer *appLayer);
37 void transmitter(LinkLayer *linkLayer);
38 void receiver(LinkLayer *linkLayer);
39 void send(LinkLayer *linkLayer);
40 int sendControl(LinkLayer *linkLayer, ControlPacket *controlPacket, int
     nParameters);
41 int sendData(LinkLayer *linkLayer, char *buffer, int size, int sequenceNumber);
42 void receive(LinkLayer *linkLayer);
43 #endif
```

11.2 appLayer.c

```
1 #include "appLayer.h"
2
3 void startAppLayer(LinkLayer *linkLayer, ApplicationLayer *appLayer)
4 {
5   if (openPort(linkLayer) < 0)
6     exit(-1);
7
8   else if (setTermiosStructure(linkLayer) < 0)
```

```
9       exit(-1);
10
11    switch (appLayer->status)
12    {
13    case TRANSMITTER:
14      transmitter(linkLayer);
15      break;
16    case RECEIVER:
17      receiver(linkLayer);
18      break;
19    }
20  }
21
22  void transmitter(LinkLayer *linkLayer)
23  {
24    if (llopenT(linkLayer) < 0)
25      exit(-1);
26
27    send(linkLayer);
28
29    if (llcloseT(linkLayer) < 0)
30      exit(-1);
31  }
32
33  void receiver(LinkLayer *linkLayer)
34  {
35    if (llopenR(linkLayer) < 0)
36      exit(-1);
37
38    receive(linkLayer);
39
40    if (llcloseR(linkLayer) < 0)
41      exit(-1);
42  }
43
44  void send(LinkLayer *linkLayer)
45  {
46    char sizeString[16];
47
48    //Start control packet
49    ControlPacket startCP;
50    startCP.controlField = 2;
51
52    //control packet name
53    TLV startTLVName;
54
55    startTLVName.type = 1;
56    startTLVName.lenght = strlen(linkLayer->fileName);
57    startTLVName.value = linkLayer->fileName;
58
59    //control packet size
60    TLV startTLVSize;
61
62    linkLayer->fileSize = getFileSize(linkLayer->fileName);
63
64    sprintf(sizeString, "%d", linkLayer->fileSize);
65
66    startTLVSize.type = 0;
67    startTLVSize.lenght = strlen(sizeString);
```

```
68    startTLVSize.value = sizeString;
69
70    TLV listParameters[2] = {startTLVSize, startTLVName};
71    startCP.parameters = listParameters;
72
73    struct timeval start;
74    gettimeofday(&start, NULL);
75
76    sendControl(linkLayer, &startCP, 2);
77    printf("Start control packet sent\n");
78
79    //Data packet
80
81    FILE *file = openFile(1, linkLayer->fileName);
82
83    char *fileData = (char *)malloc(linkLayer->fileSize);
84    int nBytesRead = 0, sequenceNumber = 0;
85
86    printf("Sending data\n");
87    while ((nBytesRead = fread(fileData, sizeof(unsigned char), BYTESTOSEND, file
        )) > 0)
88    {
89      sendData(linkLayer, fileData, BYTESTOSEND, sequenceNumber++ % 255);
90
91      memset(fileData, 0, BYTESTOSEND);
92    }
93
94    free(fileData);
95    closeFile(file);
96
97    //End control packet
98    ControlPacket endCP;
99    endCP = startCP;
100   endCP.controlField = 3;
101
102   tcflush(linkLayer->fd, TCIOFLUSH);
103
104   sendControl(linkLayer, &endCP, 2);
105   printf("End control packet sent\n");
106
107   struct timeval end;
108   gettimeofday(&end, NULL);
109
110
111   linkLayer->totalTime = (double) (end.tv_usec - start.tv_usec) / 1000000 + (
        double) (end.tv_sec - start.tv_sec);
112 }
113
114 int sendControl(LinkLayer *linkLayer, ControlPacket *controlPacket, int
        nParameters)
115 {
116   unsigned int i, packetSize = 1, index = 1;
117
118   for (i = 0; i < nParameters; i++)
119     packetSize += 2 + controlPacket->parameters[i].lenght;
120
121   unsigned char frame[packetSize];
122   frame[0] = controlPacket->controlField;
123
```

```
124   for (i = 0; i < nParameters; i++)
125   {
126     frame[index++] = controlPacket->parameters[i].type;
127     frame[index++] = controlPacket->parameters[i].lenght;
128     memcpy(&frame[index], controlPacket->parameters[i].value, controlPacket->
      parameters[i].lenght);
129     index += controlPacket->parameters[i].lenght;
130   }
131
132   if (llwrite(linkLayer, frame, packetSize) < 0)
133     exit(-1);
134
135   linkLayer->sequenceNumber = !linkLayer->sequenceNumber;
136
137   return 0;
138 }
139
140 int sendData(LinkLayer *linkLayer, char *buffer, int size, int sequenceNumber)
141 {
142   unsigned char L1, L2;
143   unsigned int packetSize;
144
145   //usando teorema do resto
146   L1 = size % 256;
147   L2 = size / 256;
148
149   packetSize = 4 + size;
150   unsigned char *frame = malloc(packetSize);
151
152   frame[0] = 1;
153   frame[1] = sequenceNumber;
154   frame[2] = L2;
155   frame[3] = L1;
156
157   memcpy(&frame[4], buffer, size);
158
159   if (llwrite(linkLayer, frame, packetSize) < 0)
160     exit(-1);
161
162   linkLayer->sequenceNumber = !linkLayer->sequenceNumber;
163
164   return 0;
165 }
166
167 void receive(LinkLayer *linkLayer)
168 {
169   int size;
170   unsigned int fileSize, index = 0;
171   char *fileName;
172   struct timeval start;
173   gettimeofday(&start, NULL);
174   //Start control packet
175
176   do
177   {
178     size = llread(linkLayer);
179   } while(linkLayer->frame[4] != 2);
180
181   while (index < size)
```

```
182   {
183     unsigned int type = linkLayer->frame[index++];  //0 = size , 1 = name
184     unsigned char lenght = linkLayer->frame[index++]; //size of file
185     char *value = malloc(lenght);           // either size or name, according
      to type
186
187     memcpy(value, &linkLayer->frame[index], lenght);
188
189     if (type == 0) //stores size of file in fileSize
190     {
191       fileSize = atoi(value);
192     }
193
194     else if (type == 1) //stores name of file in fileNAme
195     {
196       fileName = malloc(lenght);
197       memcpy(fileName, value, lenght);
198     }
199
200     index += lenght;
201   }
202
203   printf("Received start control packet.\n");
204
205   //DAQUI PARA BAIXO LE OS DADOS ATE RECEBER CONTROL PACKET A INDICAR FIM
206
207   linkLayer->fileName = fileName;
208   linkLayer->fileSize = fileSize;
209
210   FILE *file = openFile(0, linkLayer->fileName);
211
212   printf("Receiving data\n");
213
214   while (1)
215   {
216     char *data;
217     int dataC;
218     unsigned int L1, L2, lenght;
219
220     size = llread(linkLayer);
221
222     if (size < 0)
223       continue;
224
225     // frame[0] = FLAG
226     // frame[1] = A
227     // frame[2] = C da trama, nao dos dados
228     // frame[3] = BBC1
229     // A partir do 4 comeca dos dados
230     dataC = linkLayer->frame[4];
231
232     if (dataC == 3) //receives end control packet
233     {
234       printf("Received end control packet.\n");
235       break;
236     }
237
238     else if (dataC != 1)
239     {
```

```
240         linkLayer->nREJ++;
241         if (linkLayer->sequenceNumber)
242           sendMessage(linkLayer->fd, REJ1);
243         else
244           sendMessage(linkLayer->fd, REJ0);
245 
246         continue;
247       }
248 
249       L2 = linkLayer->frame[6];
250       L1 = linkLayer->frame[7];
251 
252       lenght = 256 * L2 + L1;
253 
254       data = malloc(lenght);
255 
256       memcpy(data, &linkLayer->frame[8], lenght);
257 
258       fwrite(data, sizeof(char), lenght, file);
259       free(data);
260 
261     }
262 
263     closeFile(file);
264 
265     struct timeval end;
266     gettimeofday(&end, NULL);
267 
268     linkLayer->totalTime = (double) (end.tv_usec - start.tv_usec) / 1000000 + (
        double) (end.tv_sec - start.tv_sec);
269 }
```

11.3 linkLayer.h

```
1  #ifndef LINKLAYER_H
2  #define LINKLAYER_H
3  
4  #include <stdlib.h>
5  #include <fcntl.h>
6  #include <sys/stat.h>
7  #include <sys/types.h>
8  #include <sys/time.h>
9  #include <termios.h>
10 #include <strings.h>
11 #include <signal.h>
12 #include <time.h>
13 #include "utilities.h"
14 
15 #define BAUDRATE B38400
16 
17 typedef struct
18 {
19   int fd;               /*Descritor de ficheiro */
20   char *port;             /*Dispositivo /dev/ttySx, x = 0, 1*/
21   int baudRate;           /*Velocidade de transmissao*/
22   unsigned int sequenceNumber;   /*Numero de sequencia da trama: 0, 1*/
23   unsigned int timeout;       /*Valor do temporizador:*/
24   unsigned int numTransmissions; /*Numero de tentativas em caso de falha*/
25   unsigned char *frame;       /*Trama */
```

```
26
27    char *fileName; /*nome do ficheiro */
28    int fileSize; /*tamanho do ficheiro */
29
30    unsigned int nRR; /*numero de RRs enviados/recebidos */
31    unsigned int nREJ; /*numero de REJs enviados/recebidos */
32    double totalTime; /*tempo decorrido durante a transmissao do ficheiro */
33  } LinkLayer;
34
35  struct termios oldtio, newtio;
36
37  void setupLinkLayer(LinkLayer *linkLayer,int brate,char* port,char* filename);
38  int openPort(LinkLayer *linkLayer);
39  int setTermiosStructure(LinkLayer *linkLayer);
40  int llopenT(LinkLayer *linkLayer);
41  int llopenR(LinkLayer *linkLayer);
42  int llwrite(LinkLayer *linkLayer, unsigned char *buffer, int lenght);
43  int llread(LinkLayer *linkLayer);
44  int llcloseT(LinkLayer *linkLayer);
45  int llcloseR(LinkLayer *linkLayer);
46
47  #endif
```

11.4 linkLayer.c

```
1
2  #include "linkLayer.h"
3  #include <stdio.h>
4
5  void setupLinkLayer(LinkLayer *linkLayer, int baudrate, char *port, char *
       filename)
6  {
7      linkLayer->fd = -1;
8      linkLayer->port = port;
9      linkLayer->baudRate = validBaudRate(baudrate);
10     linkLayer->sequenceNumber = 0;
11     linkLayer->timeout = 3;
12     linkLayer->numTransmissions = 3;
13     linkLayer->frame = malloc(MAX_SIZE);
14
15     linkLayer->fileName = filename;
16     linkLayer->fileSize = 0;
17
18     linkLayer->nRR = 0;
19     linkLayer->nREJ = 0;
20     linkLayer->totalTime = 0;
21 }
22
23 int openPort(LinkLayer *linkLayer)
24 {
25     linkLayer->fd = open(linkLayer->port, O_RDWR | O_NOCTTY);
26
27     if (linkLayer->fd == -1)
28     {
29         perror("openPort");
30         return -1;
31     }
32
33     return 0;
```

```
34 }
35
36 int setTermiosStructure(LinkLayer *linkLayer)
37 {
38     if (tcgetattr(linkLayer->fd, &oldtio) == -1)
39     { /* save current port settings */
40         perror("setTermiosStructure");
41         return -1;
42     }
43
44     bzero(&newtio, sizeof(newtio));
45     newtio.c_cflag = linkLayer->baudRate | CS8 | CLOCAL | CREAD;
46     newtio.c_iflag = IGNPAR;
47     newtio.c_oflag = 0;
48
49     newtio.c_lflag = 0;
50
51     newtio.c_cc[VTIME] = 1; /* inter-character timer unused */
52     newtio.c_cc[VMIN] = 0;  /* blocking read until 5 chars received */
53
54     tcflush(linkLayer->fd, TCIOFLUSH);
55
56     if (tcsetattr(linkLayer->fd, TCSANOW, &newtio) == -1)
57     {
58         perror("tcsetattr");
59         return -1;
60     }
61
62     return 0;
63 }
64
65 int llopenT(LinkLayer *linkLayer)
66 {
67
68     (void)signal(SIGALRM, alrmHanler);
69
70     while (!outOfTries(linkLayer->numTransmissions) && getTimeOut() == TRUE)
71     {
72         setTimeOut(FALSE);
73
74         alarm(linkLayer->timeout);
75
76         sendMessage(linkLayer->fd, SETUP);
77         validateCommand(linkLayer->fd, UA);
78
79         alarm(0);
80     }
81
82     if (outOfTries(linkLayer->numTransmissions))
83     {
84         printf("Failed to send the message (%d attemps)\n", linkLayer->
    numTransmissions);
85         return -1;
86     }
87
88     return 0;
89 }
90
91 int llopenR(LinkLayer *linkLayer)
```

```
92  {
93      setTimeOut(FALSE);
94
95      validateCommand(linkLayer->fd, SETUP);
96      sendMessage(linkLayer->fd, UA);
97
98      return 0;
99  }
100
101 int llwrite(LinkLayer *linkLayer, unsigned char *buffer, int lenght)
102 {
103
104     unsigned char *packet = malloc(12000);
105     resetTries();
106     setTimeOut(TRUE);
107
108     packet[0] = FLAG;
109     packet[1] = A;
110     packet[2] = linkLayer->sequenceNumber << 6;
111     packet[3] = packet[1] ^ packet[2];
112
113     memcpy(&packet[4], buffer, lenght);
114
115     unsigned char BCC2 = 0;
116     int i = 0;
117
118     for (; i < lenght; i++)
119         BCC2 ^= buffer[i];
120
121     packet[lenght + 4] = BCC2;
122     packet[lenght + 5] = FLAG;
123
124     int newLenght = stuffing(packet, lenght + 6);
125
126     while (!outOfTries(linkLayer->numTransmissions) && getTimeOut() == TRUE)
127     {
128         setTimeOut(FALSE);
129         alarm(linkLayer->timeout);
130
131         if (write(linkLayer->fd, packet, newLenght + 6) < 0)
132         {
133             perror("write");
134             return -1;
135         }
136
137         unsigned char response = 10;
138         response = receiveResponse(linkLayer->fd);
139
140         if (response == C_RR0 ||
141             response == C_RR1)
142             linkLayer->nRR++;
143
144         else if (response == C_REJ0 ||
145                  response == C_REJ1)
146         {
147             setTimeOut(TRUE);
148             incTries();
149             linkLayer->nREJ++;
150         }
```

```
151
152         alarm(0);
153     }
154
155     if (outOfTries(linkLayer->numTransmissions))
156     {
157         printf("Failed to send the message (%d attemps)\n", linkLayer->
    numTransmissions);
158         return -1;
159     }
160
161     return 0;
162 }
163
164 int llread(LinkLayer *linkLayer)
165 {
166     setTimeOut(FALSE);
167
168     int size = validateFrame(linkLayer->fd, linkLayer->frame);
169
170     size = destuffing(linkLayer->frame, size);
171
172     if (linkLayer->sequenceNumber != linkLayer->frame[2] >> 6 ||
173         !isValidBcc2(linkLayer->frame, size, linkLayer->frame[size - 2]))
174     {
175         linkLayer->nREJ++;
176         size = -1;
177
178         if (linkLayer->sequenceNumber)
179             sendMessage(linkLayer->fd, REJ1);
180         else
181             sendMessage(linkLayer->fd, REJ0);
182     }
183
184     else
185     {
186         linkLayer->nRR++;
187
188         if (linkLayer->sequenceNumber)
189             sendMessage(linkLayer->fd, RR1);
190         else
191             sendMessage(linkLayer->fd, RR0);
192
193         linkLayer->sequenceNumber = !linkLayer->sequenceNumber;
194     }
195
196     return size;
197 }
198
199 int llcloseT(LinkLayer *linkLayer)
200 {
201     resetTries();
202     setTimeOut(TRUE);
203
204     while (!outOfTries(linkLayer->numTransmissions) && getTimeOut() == TRUE)
205     {
206
207         setTimeOut(FALSE);
208         alarm(linkLayer->timeout);
```

```
209
210         sendMessage(linkLayer->fd, DISC);
211
212         validateCommand(linkLayer->fd, DISC);
213
214         alarm(0);
215     }
216
217     if (outOfTries(linkLayer->numTransmissions))
218     {
219         printf("Failed to send the message (%d attemps)\n", linkLayer->
    numTransmissions);
220         return -1;
221     }
222
223     sendMessage(linkLayer->fd, UA);
224
225     return 0;
226 }
227
228 int llcloseR(LinkLayer *linkLayer)
229 {
230     resetTries();
231     setTimeOut(TRUE);
232
233     (void)signal(SIGALRM, alrmHanler);
234
235     validateCommand(linkLayer->fd, DISC);
236
237     while (!outOfTries(linkLayer->numTransmissions) && getTimeOut() == TRUE)
238     {
239
240         setTimeOut(FALSE);
241         alarm(linkLayer->timeout);
242
243         sendMessage(linkLayer->fd, DISC);
244
245         validateCommand(linkLayer->fd, UA);
246
247         alarm(0);
248     }
249
250     if (outOfTries(linkLayer->numTransmissions))
251     {
252         printf("Failed to send the message (%d attemps)\n", linkLayer->
    numTransmissions);
253         return -1;
254     }
255
256     return 0;
257 }
```

11.5 main.c

```
1
2 #include "linkLayer.h"
3 #include "appLayer.h"
4 #include <stdlib.h>
5 #include <stdio.h>
```

```
#include <string.h>
#include <time.h>

void printStats (LinkLayer * linkLayer)
{
  printf("\nFile name = %s\n", linkLayer->fileName);
  printf("File size = %d\n", linkLayer->fileSize);
  printf("#RR = %d \n#REJ = %d\n", linkLayer->nRR, linkLayer->nREJ);
  printf("Time: %f s\n\n", linkLayer->totalTime);
}




int  printChose()
{
  int bChose;
  printf("Chose the Baudrate to transmit the File:");
  printf("\n");
  printf("1->600   2->1200  3->1800   4->2400 ");
  printf("\n");
  printf("5->4800  6->9600  7->19200  8->38400 ");
  printf("\n\n");

  printf("Enter the value:");
  scanf("%d", &bChose);

  while(bChose<1 || bChose>8){
    printf("Invalid value\n");
    printf("Enter the value:");
    scanf("%d", &bChose);

  }
  return bChose;
}

int main(int argc, char *argv[]) {

  srand ( time(NULL) );
  int resultB;


  LinkLayer * linkLayer = malloc(sizeof(LinkLayer));


  if ( (argc != 4) ||
       ((strcmp("/dev/ttyS0", argv[1])!=0) &&
        (strcmp("/dev/ttyS1", argv[1])!=0) )) {
    printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS0 mode filename
   \n");
    exit(1);
   }


  resultB=printChose();
  setupLinkLayer (linkLayer,resultB,argv[1],argv[3]);

  ApplicationLayer * appLayer = malloc(sizeof(ApplicationLayer));

```

```
64   if ((strcmp("0", argv[2]) == 0)){
65     appLayer->status = TRANSMITTER;
66
67   }
68   else if ((strcmp("1", argv[2]) == 0))
69     appLayer->status = RECEIVER;
70   else
71   {
72     printf("Mode must be either 0 (TRANSMITTER) or 1 (RECEIVER)\n.");
73     return -1;
74   }
75
76   startAppLayer(linkLayer, appLayer);
77   printStats(linkLayer);
78   return 0;
79 }
```

11.6 utilities.h

```
1  #ifndef UTILITIES_H
2  #define UTILITIES_H
3
4  #include <sys/types.h>
5  #include <sys/stat.h>
6  #include <fcntl.h>
7  #include <termios.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <unistd.h>
11 #include <string.h>
12 #include <signal.h>
13
14 #define MAX_SIZE 255
15 #define BAUDRATE B38400
16 #define MODEMDEVICE "/dev/ttyS1"
17 #define ERRORPROBABILITY1 0
18 #define ERRORPROBABILITY2 0
19 #define _POSIX_SOURCE 1 /* POSIX compliant source */
20 #define FALSE 0
21 #define TRUE 1
22
23 #define FLAG 0x7E
24 #define ESC 0x7D
25 #define FLAG_S_2 0x5E
26 #define FLAG_S_3 0x5D
27
28 #define C_REJ0 0X01
29 #define C_REJ1 0X81
30
31 #define C_RR0 0X05
32 #define C_RR1 0X85
33
34 #define A 0x03
35 #define C_SET 0x03
36 #define C_DISC 0x0B
37 #define C_UA 0x07
38 #define BYTE_TO_SEND 5
39 #define FINALSTATE 5
40
```

```
41 static const unsigned char SETUP[5] = { FLAG, A, C_SET, A ^ C_SET, FLAG };
42 static const unsigned char UA[5] = { FLAG, A, C_UA, A ^ C_UA, FLAG };
43 static const unsigned char DISC[5] = { FLAG, A, C_DISC, A ^ C_DISC, FLAG };
44 static const unsigned char RR0[5] = { FLAG, A, C_RR0, A ^ C_RR0, FLAG };
45 static const unsigned char RR1[5] = { FLAG, A, C_RR1, A ^ C_RR1, FLAG };
46 static const unsigned char REJ0[5] = { FLAG, A, C_REJ0, A ^ C_REJ0, FLAG };
47 static const unsigned char REJ1[5] = { FLAG, A, C_REJ1, A ^ C_REJ1, FLAG };
48
49
50
51 void alrmHanler(int sig);
52 void resetTries();
53 void incTries();
54 int outOfTries (int maxTries);
55 void setTimeOut (int value);
56 int getTimeOut ();
57 FILE* openFile(int type ,char* filePath);
58 void closeFile(FILE * file);
59 unsigned int getFileSize(char *fileName);
60 void sendMessage(int fd, const unsigned char cmd[]);
61 int validateCommand(int fd, const unsigned char cmd[]);
62 int validateFrame(int fd, unsigned char * frame);
63 char receiveResponse(int fd);
64 int stuffing(unsigned char* frame,int size);
65 int destuffing(unsigned char* frame,int size);
66 int isValidBcc2(unsigned char * packet,int packetSize,unsigned char received);
67 int simulateError();
68 int  validBaudRate(int brate);
69
70 #endif
```

11.7 utilities.h

```
1
2 #include "utilities.h"
3
4 volatile int STOP = FALSE;
5 int CANCEL = FALSE;
6
7 int timeOut = TRUE;
8 int tries = 0;
9
10 void alrmHanler(int sig)
11 {
12   printf("alarme # %d\n", tries + 1);
13   timeOut = TRUE;
14   tries++;
15 }
16
17 void resetTries() { tries = 0; }
18
19 void incTries() { tries++; }
20
21 int outOfTries(int maxTries) { return tries >= maxTries; }
22
23 void setTimeOut(int value)
24 {
25   timeOut = value;
26 }
```

```
27
28 int getTimeOut() { return timeOut; }
29
30 FILE *openFile(int type, char *filePath)
31 {
32   FILE *result;
33
34   if (type == 0)
35     result = fopen(filePath, "wb");
36
37   else
38     result = fopen(filePath, "rb");
39
40   if (result == NULL)
41   {
42     perror("error to open the file ");
43     exit(-1);
44   }
45   return result;
46 }
47
48 void closeFile(FILE *file)
49 {
50   if (fclose(file) != 0)
51   {
52     perror("closeFile");
53     exit(-1);
54   }
55 }
56
57 unsigned int getFileSize(char *fileName)
58 {
59   struct stat st;
60   if (stat(fileName, &st) < 0)
61   {
62     perror("getFileSize");
63     exit(-1);
64   }
65
66   return st.st_size;
67 }
68
69 void sendMessage(int fd, const unsigned char cmd[])
70 {
71   int byteChar = 0;
72
73   while (byteChar != BYTE_TO_SEND)
74   {
75
76     byteChar = write(fd, cmd, BYTE_TO_SEND);
77
78     if (byteChar == -1)
79     {
80       perror("sendMessage");
81       exit(-1);
82     }
83   }
84 }
85
```

```
86 int validateCommand(int fd, const unsigned char cmd[])
87 {
88   int state = 0, aux;
89   unsigned char reader;
90 
91   while (state != FINALSTATE && timeOut == FALSE)
92   {
93 
94     aux = read(fd, &reader, 1);
95 
96     if (aux == -1)
97     {
98       perror("validateCommand");
99       exit(-1);
100    }
101 
102    switch (state)
103    {
104    case 0:
105      if (reader == cmd[0])
106      {
107        state = 1;
108      }
109      else
110        state = 0;
111      break;
112    case 1:
113      if (reader == cmd[1])
114        state = 2;
115 
116      else if (reader != cmd[0])
117        state = 0;
118      break;
119    case 2:
120      if (reader == cmd[2])
121        state = 3;
122 
123      else if (reader != cmd[0])
124        state = 0;
125      break;
126    case 3:
127 
128      if ((cmd[3]) == reader)
129        state = 4;
130      else
131        state = 0;
132      break;
133    case 4:
134      if (reader == cmd[4])
135        state = 5;
136 
137      else
138        state = 0;
139      break;
140    }
141  }
142  return 0;
143 }
144 
```

```
145 int validateFrame(int fd, unsigned char *frame)
146 {
147
148   int state = 0, bytesRead, dataSize = 0;
149   unsigned char reader;
150   while (state != FINALSTATE && timeOut == FALSE)
151   {
152     bytesRead = read(fd, &reader, 1);
153
154     if (bytesRead == -1)
155     {
156       perror("validateFrame");
157       exit(-1);
158     }
159
160     switch (state)
161     {
162
163     case 0: //start
164       if (reader == FLAG)
165       {
166         frame[0] = reader;
167         state = 1;
168       }
169       break;
170
171     case 1: //flag
172       if (reader == A)
173       {
174         frame[1] = reader;
175         state = 2;
176       }
177
178       else if (reader != FLAG)
179         state = 0;
180       break;
181
182     case 2: //A
183
184       if (reader == 0 || reader == (1 << 6))
185       {
186         frame[2] = reader;
187         state = 3;
188       }
189
190       else if (reader == FLAG)
191         state = 1;
192
193       else
194         state = 0;
195       break;
196
197     case 3: //C
198       if ((frame[1] ^ frame[2]) == reader)
199       {
200         frame[3] = reader;
201         state = 4;
202       }
203
```

```
204       else if (reader == FLAG)
205         state = 1;
206
207       else
208         state = 0;
209
210       if (simulateError(1))
211       {
212         state = 0;
213         printf("Error simulate\n");
214       }
215       break;
216
217     case 4:                 //BCC
218       frame[4 + dataSize] = reader; //Vai colocando dados ate encontrar flag
219       dataSize++;
220       if (reader == FLAG)
221         state = 5;
222
223       break;
224     }
225   }
226
227   return 4 + dataSize; // F + A + C + BCC1 + dataSize (Dados + BCC2 + FLAG2)
228 }
229
230 char receiveResponse(int fd)
231 {
232   int state = 0, aux;
233   unsigned char reader;
234   unsigned char commandReceived;
235
236   while (state != FINALSTATE && timeOut == FALSE)
237   {
238
239     aux = read(fd, &reader, 1);
240
241     if (aux == -1)
242     {
243       perror("receiveResponse");
244       exit(-1);
245     }
246
247     switch (state)
248     {
249     case 0:
250       if (reader == FLAG)
251       {
252         state = 1;
253       }
254       else
255         state = 0;
256       break;
257     case 1:
258       if (reader == A)
259         state = 2;
260
261       else if (reader != FLAG)
262         state = 0;
```

```
263       break;
264     case 2:
265       if (reader == C_RR0 ||
266         reader == C_RR1 ||
267         reader == C_REJ0 ||
268         reader == C_REJ1)
269       {
270         commandReceived = reader;
271         state = 3;
272       }
273 
274       else if (reader != FLAG)
275         state = 0;
276       break;
277     case 3:
278       if ((commandReceived ^ A) == reader)
279         state = 4;
280       else
281         state = 0;
282 
283 
284       break;
285     case 4:
286       if (reader == FLAG)
287         state = 5;
288 
289       else
290         state = 0;
291       break;
292     }
293   }
294 
295   if (state == FINALSTATE)
296     return commandReceived;
297 
298   return 0;
299 }
300 
301 int stuffing(unsigned char *frame, int size)
302 {
303   char *result = malloc(20000);
304   int resultSize = size;
305   int i;
306 
307   for (i = 1; i < (size - 1); i++)
308   {
309     if (frame[i] == FLAG || frame[i] == ESC)
310     {
311       resultSize++;
312     }
313   }
314   result[0] = frame[0];
315   int j = 1;
316 
317   for (i = 1; i < (size - 1); i++)
318   {
319     if (frame[i] == FLAG || frame[i] == ESC)
320     {
321       result[j] = ESC;
```

```
322       result[++j] = frame[i] ^ 0X20;
323     }
324     else
325     {
326       result[j] = frame[i];
327     }
328     j++;
329   }
330
331   result[j] = frame[i];
332   frame = realloc(frame, size + resultSize - size);
333   memcpy(frame, result, resultSize);
334   return resultSize;
335 }
336
337 int destuffing(unsigned char *frame, int size)
338 {
339   int i, j = 0;
340   char *result = malloc(20000);
341   int resultSize = size;
342
343   for (i = 1; i < (size - 1); i++)
344   {
345     if (frame[i] == ESC)
346       resultSize--;
347   }
348
349   for (i = 0; i < size; i++)
350   {
351     if (frame[i] == ESC)
352       result[j] = frame[++i] ^ 0X20;
353
354     else
355       result[j] = frame[i];
356
357     j++;
358   }
359   memcpy(frame, result, resultSize);
360
361   return resultSize;
362 }
363
364 int isValidBcc2(unsigned char *packet, int packetSize, unsigned char received)
365 {
366   unsigned char expected = 0;
367
368   unsigned int i = 4;
369   for (; i < packetSize - 2; i++)
370   {
371     expected ^= packet[i];
372   }
373
374   if (simulateError(2))
375   {
376     printf("Error simulate2\n");
377     return 0;
378   }
379
380   return (expected == received);
```

```
}

int simulateError (int value)
{
  int n = rand() % 100;

  if (value == 1)
    return n < ERRORPROBABILITY1;
  if (value == 2)
    return n < ERRORPROBABILITY2;

  return 0;
}

int validBaudRate(int brate){
  int speed;
  switch(brate){
  case 1: speed=B600;
          break;
  case 2: speed=B1200;
          break;
  case 3: speed=B1800;
          break;
  case 4: speed=B2400;
          break;
  case 5: speed=B4800;
          break;
  case 6: speed=B9600;
          break;
  case 7: speed=B19200;
          break;
  case 8: speed=B38400;
          break;

  }
return speed;

}
```