Protocolo de Ligação de Dados

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Turma 6:
Duarte Miguel de Novo Faria - 201607176
Maria Teresa Queiroz Machado Urbano Ferreira - 201603811
Maria João Senra Viana - 201604751

Faculdade de Engenharia da Universidade do Porto

12 de Novembro de 2018

Conteúdo

1 Sumário **3**

2 Introdução **3**

3 Arquitetura **3**

4 Estrutura do código **4**
4.1 *Link Layer* . . . 4
4.2 *Application Layer* . . . 4

5 Casos de uso principais **4**

6 Protocolo de Ligação Lógica **5**
6.1 llopen . . . 5
6.2 llwrite . . . 5
6.3 llread . . . 5
6.4 llclose . . . 5

7 Protocolo de Aplicação **6**
7.1 set_connection . . . 6
7.2 transmitterMode . . . 6
7.3 receiverMode . . . 6

8 Validação **6**

9 Eficiência do protocolo de ligação de dados **7**
9.1 Variação do tamanho dos pacotes . . . 7
9.2 Variação da capacidade de ligação . . . 7
9.3 Variação do FER . . . 8
9.4 Variação do T_prop . . . 8
9.5 Variação da eficácia . . . 9

10 Conclusão **9**

A Código fonte **10**
A.1 applicationLayer.h . . . 10
A.2 applicationLayer.c . . . 11
A.3 linkLayer.h . . . 15
A.4 linkLayer.c . . . 16
A.5 utilities.h . . . 26
A.6 makefile . . . 27

1 Sumário

Este projeto tem como objetivo a implementação de uma aplicação simples que permitisse a transferência de ficheiros entre dois computadores ligados fisicamente por uma porta série através de uma comunicação assíncrona. Deste modo, foram implementadas um conjunto de funções de leitura, escrita e tratamento de dados.

O projeto foi concluído com sucesso. Foi desenvolvida uma aplicação capaz de enviar e receber dados corretamente, sendo que quando ocorre uma falha na transmissão de dados, o programa é capaz de restabelecer a transmissão, tratando dos erros devidamente.

2 Introdução

Os objetivos propostos para este primeiro trabalho laboratorial consistem em implementar um protocolo de ligação de dados, de acordo com a especficação fornecida e testar esse protocolo com uma aplicação simples de transferância de ficheiros, igualmente especificada. Este relatório encontra-se dividido em diversas secções retratando diferentes partes do projeto:

- Arquitetura, onde são apresentados os blocos funcionais e as interfaces.
- Estrutura do Código, referindo as APIs implementadas e as principais estruturas de dados.
- Casos de Uso Principais, fazendo a sua identificação e apresentando as sequências de chamada de funções.
- Protocolo de Ligação Lógica, onde se identificam as estratégias de implementação do mesmo, com referências oportunas ao código.
- Protocolo de Apicação, onde se faz uma descrição análoga à do outro protocolo.
- Validação, onde são desscritos os testes efetuados de modo a verificar o correto funcionamento do programa.
- Eficiência do protocolo de ligação ede dados, onde se caracteriza a estatística da eficiência do protocolo, feito com recurso a medidas sobre o código desenvolvido. A caracterização teórica de um protocolo Stop&Wait, que deverá ser usada como termo de comparação, encontra-se descrita nos slides de Ligação Lógica das aulas teóricas.
- Conclusões, elaborando uma síntese das secções apresentadas e uma breve reflexão sobre os objetivos alcançados.

3 Arquitetura

O trabalho está dividido em duas camadas, a Application Layer e a Link Layer. Esta estrutura em camadas permite que as camadas sejam funcionalmente invisíveis entre si, ou seja, não conhecem os detalhes inerentes a cada uma.

4 Estrutura do código

Existem duas principais structs onde são guardadas as informações iniciais provenientes do utilizador e outras que importem manter ao longo da execução do programa: *Application Layer* e *Link Layer*.

4.1 *Link Layer*

É representada por uma struct onde são guardados os dados relativos à mesma. É constituída pelo descritor correspondente à porta série, velocidade de transmissão, número de sequência de trama, valor do temporizador, número de tentativas em caso de falha, modo de transmissão e finalmente por uma struct onde são guardadas as definições da porta de série. Nesta camada existem quatro funções principais:

- llopen, responsável por estabelecer a a conexão inicial entre o recetor e transmissor enviando as tramas SET e UA.
- llread
- llwrite
- llclose, onde se encontram os aspetos de terminação, como o envio das tramas DISC e UA.

4.2 *Application Layer*

Tal como a Link Layer, é também representada por uma struct que inclui o descritor correspondente à porta série, modo de transmissão e ainda um inteiro que diferencia se estamos a usar a porta ttyS0 ou ttyS1. As principais funções são:

- set_connection, que inicializa a struct para tudo poder funcionar.
- transmitterMode, que envia os pacotes de dados com informação.
- receiverMode, que recebe os pacotes de dados com informação.

5 Casos de uso principais

A nossa aplicação necessita de três parâmetros para correr sendo que um deles é a porta série, o outro um caracter (T ou R) para indicar se queremos correr o programa como transmissor ou recetor e por último o nome do ficheiro a enviar. Depois disso, é inicializada a struct da Link Layer, seguindo-se a invocação da função set_connection que por sua vez invoca a função llopen. A transmissão de dados dá-se com a seguinte sequência: configuração da ligação entre os dois computadores, estabelecimento da ligação, transmissor envia dados, recetor recebe dados, recetor guarda dados num ficheiro com o mesmo nome do ficheiro enviado pelo emissor, terminação da ligação.

6 Protocolo de Ligação Lógica

Esta camada é responsável por:

- Configurar a porta série.
- Estabelecimento e terminação da ligação através da porta série.
- Envio e receção de mensagens e de comandos.
- Stuff e destuff dos pacotes da camada da aplicação.

6.1 llopen

Esta função tem a responsabilidade de estabelecer a ligação através da porta série em que inicialmente chama a função setTermios que configura a porta série com as especificações pretendidas. Envia o SET pelo emissor seguido de resposta UA pelo recetor. Se UA não for recebido no tempo definido por timeOut(s), o SET é reenviado e aguarda-se pelo UA, repetindo-se este processo n vezes em que n é transmissions. Se este n for excedido, a aplicação acaba com uma mensagem de erro.

6.2 llwrite

Esta função inicialmente cria as tramas de informação e escreve na porta série o packet que recebe e aguarda uma resposta. Se não a receber, vai voltar a tentar o número de vezes definido podendo receber dois comandos: RR e REJ. Se receber o primeiro, vai retornar terminando com sucesso; se receber o segundo, quer dizer que foi rejeitada e vai ser retransmitida até atingir o número máximo de tentativas definido.

6.3 llread

Esta é a função responsável pela receção das tramas e pelo destuffing das mesmas.É verificado o BCC2 e, caso este esteja correto, é enviado o RR, caso contrário é enviado o REJ, sendo que o campo de controlo é enviado dependentemente do número de sequência da trama (sequenceNumber).

6.4 llclose

A função llclose é responsável por terminar a ligação entre o emissor e o recetor. É enviado o comando DISC e aguarda pela sua receção. Neste caso, envia o UA, terminando assim a aplicação, caso contrário aborta.

7 Protocolo de Aplicação

Esta camada é responsável por:

- Envio dos pacotes de controlo START e END.
- Divisão do ficheiro em fragmentos quando se trata do emissor e a concatenação dos fragmentos recebidos, quando se trata do recetor.
- Leitura do ficheiro a enviar e criação do ficheiro.

7.1 set_connection

É nesta função que se inicia a struct para o programa começar a funcionar, invocando a função llopen.

7.2 transmitterMode

Nesta função são criados e enviados os pacotes de controlo START e END tal como os pacotes de dados.

7.3 receiverMode

Nesta função são recebidos os pacotes de controlo e os pacotes de dados.

8 Validação

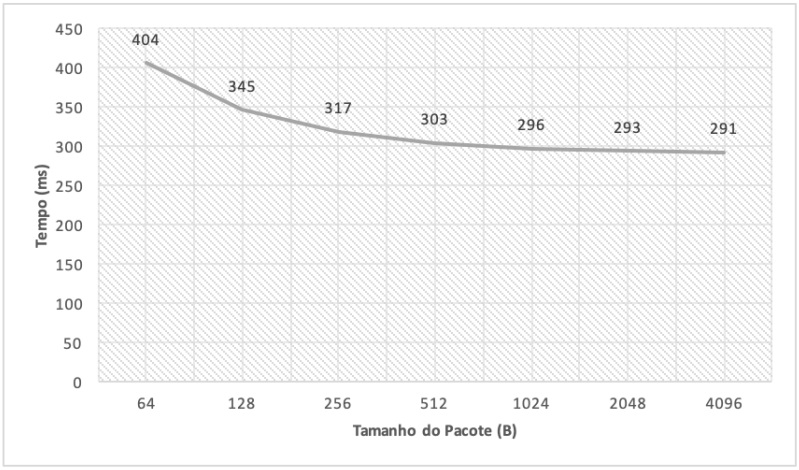
Para validar o protocolo desenvolvido foram realizados vários testes durante o desenvolvimento e demonstração do mesmo, desde uma transferência normal, transferência com interrupções momentâneas de conexão e envio esporádico de erros através de curto circuito na porta de série. Todos concluídos com sucesso.

9 Eficiência do protocolo de ligação de dados

De forma a avaliar a eficiência do protocolo desenvolvido foram feitos os testes seguintes:

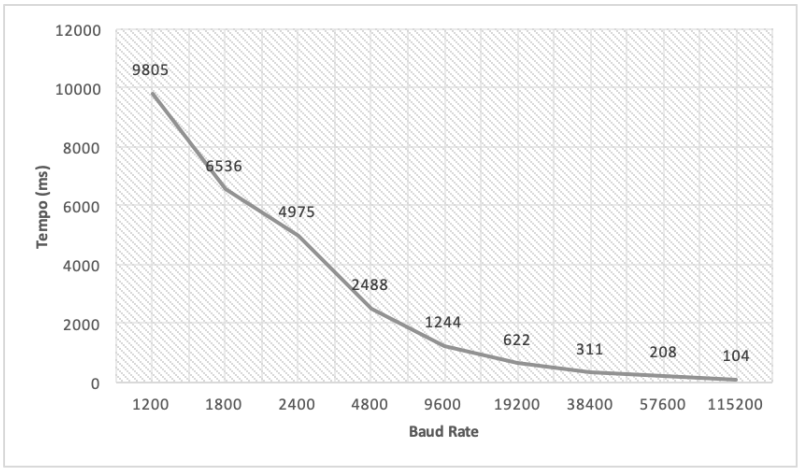
9.1 Variação do tamanho dos pacotes

Com este gráfico podemos observar que quanto maior o tamanho dos pacotes, mais eficiente é a aplicação. Isto porque é mandada mais informação de uma vez o que faz que menos tramas sejam mandadas e que o programa execute mais rapidamente.



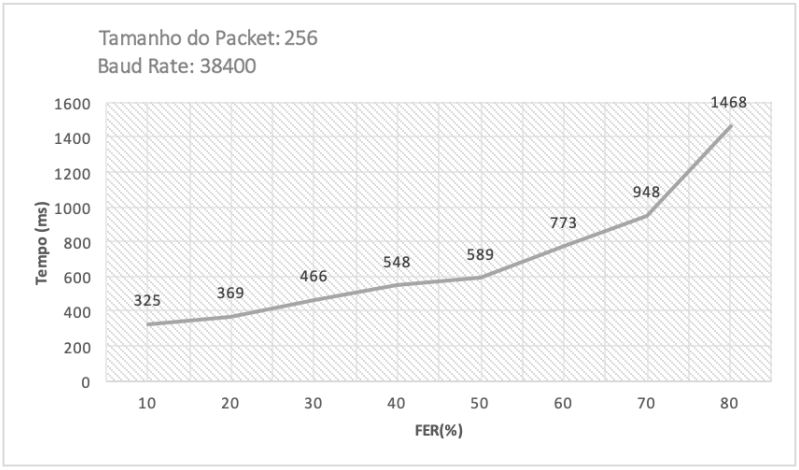
9.2 Variação da capacidade de ligação

Com este gráfico podemos concluir que com o aumento da capacidade de ligação, o programa demora menos tempo a executar.



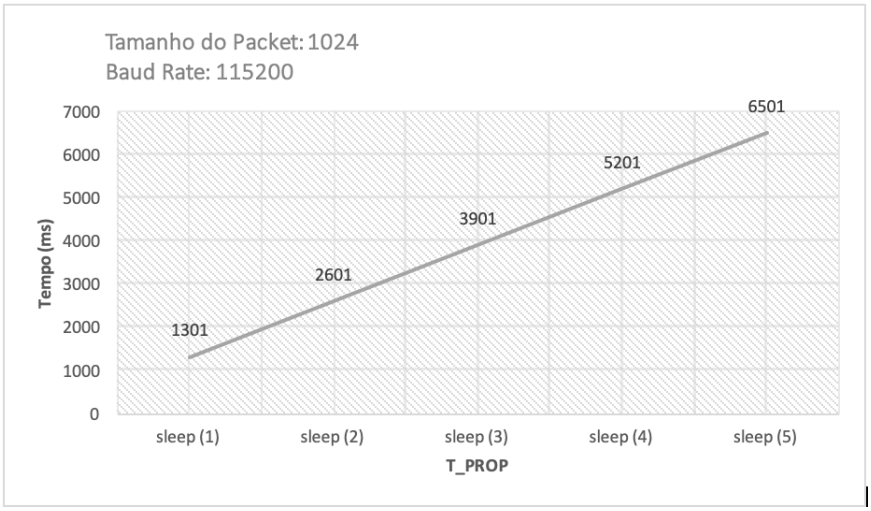
9.3 Variação do FER

Através deste gráfico conclui-se que a quantidade de erros (nos cabeçalhos ou nos campos de dados das tramas I) têm um impacto significativo no tempo de transferência.



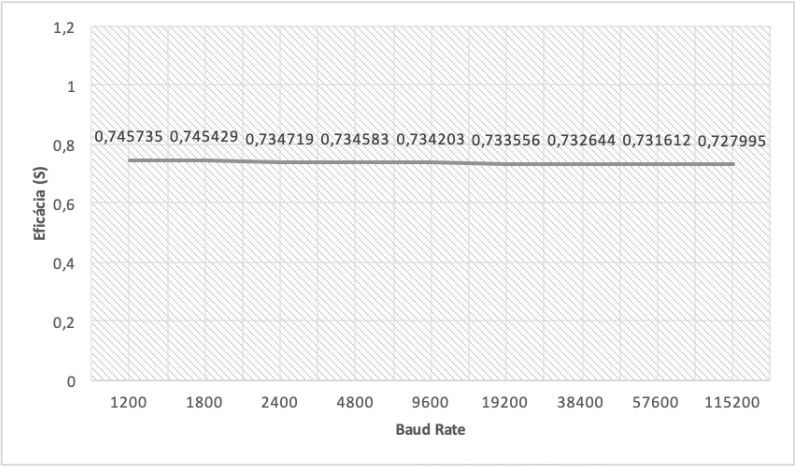
9.4 Variação do T_prop

Através deste teste, pode-se afirmar que o tempo de propagação tem um impacto linear no tempo de transferência total, ou seja, de acordo com o esperado.



9.5 Variação da eficácia

Com este gráfico concluimos que independentemente do valor da capacidade de ligação a eficácia do programa continua constante, o que é um ponto positivo.



10 Conclusão

O desenvolvimento deste projeto foi exigente devido às especificidades pretendidas, principalmente a forma de iteração e sincronização da camada de aplicação e de dados em ambos os computadores. Em suma, o trabalho foi concluído com sucessom tendo-se cumprido todos os objetivos, e a sua elaboração contribuiu positivamente para um aprofundamento do conhecimento, tanto teórico como prático, do tema em questão.

A Código fonte

A.1 applicationLayer.h

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <termios.h>
#include <fcntl.h>
#include <stdbool.h>
#include <signal.h>
#include <unistd.h>
#include <errno.h>
#include "linkLayer.h"

typedef struct  {
  int port;
  int fileDescriptor;/*Descritor correspondente a porta serie*/
  status mode;/*TRANSMITTER | RECEIVER*/
}applicationLayer;

applicationLayer app;

int createControlPacket(char* filename, unsigned long filesize, unsigned char control_byte,
    unsigned char * packet);
int createDataPacket(unsigned char* data, int dataSize, unsigned char* packet);
int receivePacket(unsigned char *packet, int fd);
void receiveDataPacket(unsigned char *packet, int fd);
void receiveControlPacket(unsigned char *packet, unsigned char control_byte);
unsigned char *readFile(char *fileName, unsigned long fileSize);
void set_connection(char * port, char * stat);
int transmitterMode(char* fileName);
int receiverMode(char* fileName);
```

applicationLayer.h

A.2 applicationLayer.c

```
#include "applicationLayer.h"

unsigned char seqNum = 0;
int fDes;

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

  if ( (argc < 4) ||
       ((strcmp(COM1_PORT, argv[1])!=0) && (strcmp(COM2_PORT, argv[1])!=0)) ||
       ((strcmp("T", argv[2]) !=0) &&  (strcmp("R", argv[2]) !=0))) {

    printf("Usage:\tnserial SerialPort ComunicationMode\n\tex: nserial /dev/ttyS1 R penguin.
        gif\n");
    exit(1);
  }

  char *filename = argv[3];

  initDataLinkStruct(TRANSMISSIONS, TIMEOUT, BAUDRATE);

  set_connection(argv[1],argv[2]);

  if(app.mode == TRANSMITTER){
      if (transmitterMode(filename) == -1)
        return -1;
  }
  else if (app.mode == RECEIVER) {
      receiverMode(filename);
  }

  llclose(app.fileDescriptor);
  return 0;
}

int transmitterMode(char* fileName) {
    int file;
    struct stat data;
    long fileSize;

    if ((file = open(fileName, O_RDONLY)) == -1) {
        perror("Error while opening the file");
        return 0;
    }

    stat((const char *)fileName, &data); //get the file metadata
    fileSize = data.st_size; //gets file size in bytes

    int rejCounter = 0;
    unsigned char startPacket[PACKET_SIZE];
    int packetSize = createControlPacket(fileName, fileSize, START, startPacket);
    llwrite(app.fileDescriptor, startPacket, packetSize, &rejCounter);

    int size;

    unsigned char msg[DATA_PACKET_SIZE];
    unsigned char packet[PACKET_SIZE];

    while ((packetSize = read(file, msg, DATA_PACKET_SIZE)) != 0) {
      size = createDataPacket(msg, packetSize, packet);
      if (llwrite(app.fileDescriptor, packet, size, &rejCounter) == -1)
        return -1;
    }

```

```
63      unsigned char endPacket[PACKET_SIZE];
64      packetSize = createControlPacket(fileName, fileSize, END, endPacket);
65      llwrite(app.fileDescriptor, endPacket, packetSize, &rejCounter);
66      return 0;
67  }
68
69  int receiverMode(char* filename) {
70      int packetSize = 0;
71      unsigned char dataPacket[(PACKET_SIZE+5)*2];
72
73      int fd = open(filename, O_WRONLY | O_TRUNC | O_CREAT | O_APPEND );
74
75      while(dataPacket[0] != 3){
76        packetSize = 0;
77        if(!llread(app.fileDescriptor, dataPacket, &packetSize)){
78          receivePacket(dataPacket, fd);
79        }
80      }
81      close(fd);
82      return 0;
83  }
84
85  int createControlPacket(char* filename, unsigned long filesize, unsigned char control_byte,
        unsigned char * packet){
86
87    packet[0] = control_byte;
88    packet[1] = FILE_SIZE;
89    packet[2] = sizeof(filesize);
90    packet[3] = (filesize >> 24) & 0xFF;
91    packet[4] = (filesize >> 16) & 0xFF;
92    packet[5] = (filesize >> 8) & 0xFF;
93    packet[6] = filesize & 0xFF;
94    packet[7] = FILE_NAME;
95    packet[8] = strlen(filename) + 1;
96    int i=0;
97    for(; i< strlen(filename) + 1; i++){
98      packet[i+9] = filename[i];
99    }
100
101   return i+9;
102 }
103
104 int createDataPacket(unsigned char* data, int dataSize, unsigned char* packet){
105   packet[0] = DATA;
106   packet[1] = seqNum++;
107   packet[2] = dataSize/256;
108   packet[3] = dataSize%256;
109   int i = 0;
110   for(; i < dataSize; i++){
111       packet[i+4] = data[i];
112   }
113   return i+4;
114 }
115
116 void receiveDataPacket(unsigned char *packet, int fd){
117   unsigned char l1, l2;
118   int k;
119   l1 = packet[3];
120   l2 = packet[2];
121   k = 256*l2 + l1;
122   unsigned char d[k];
123   int i = 0;
124  for(; i < k; i++){
125    d[i] = packet[i+4];
126  }
```

```
127
128   write(fd, d, k);
129 }
130
131 void receiveControlPacket(unsigned char *packet, unsigned char control_byte){
132   // int fileSizeLength = (int) packet[2];
133   unsigned long fileSize = 0;
134
135   fileSize += packet[6];
136   fileSize += packet[5] << 8;
137   fileSize += packet[4] << 16;
138   fileSize += packet[3] << 24;
139   printf("fileSize: %ld\n", fileSize);
140
141   int filenameSize = (int) packet[8];
142   unsigned char filename[filenameSize];
143   int i = 0;
144   for(; i < filenameSize; i++){
145     filename[i] = packet[i+9];
146   }
147
148   printf("filename: %s\n", filename);
149 }
150
151 int receivePacket(unsigned char *packet, int fd){
152   switch(packet[0]){
153     case 1:
154       receiveDataPacket(packet, fd);
155       break;
156     case 2:
157       receiveControlPacket(packet, START);
158       break;
159     case 3:
160       receiveControlPacket(packet, END);
161       break;
162     default:
163       break;
164   }
165   return 0;
166 }
167
168 void set_connection(char * port, char * stat){
169
170   if(strcmp(port,COM1_PORT)==0)
171   app.port = COM1;
172
173   if(strcmp(port,COM2_PORT)== 0)
174   app.port =COM2;
175
176   if(strcmp(stat,"T")==0){
177     app.mode = TRANSMITTER;
178   }
179
180   if(strcmp(stat,"R")== 0){
181     app.mode = RECEIVER;
182   }
183
184   app.fileDescriptor = llopen(app.port,app.mode);
185
186   if(app.fileDescriptor< 0){
187     printf("app - set_connection(): invalid file descriptor\n");
188     exit(-1);
189   }
190
191 }
```

A.3 linkLayer.h

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <termios.h>
#include <fcntl.h>
#include <stdbool.h>
#include <signal.h>
#include <unistd.h>
#include <errno.h>

#include "utilities.h"

typedef struct {
  char port[20]; //dispositivo /dev/ttySx, x = 0, 1
  int baudRate; //velocidade de transmissao
  unsigned int sequenceNumber; //numero da sequencia de trama: 0 ou 1
  unsigned int timeout; //valor do temporizador: 1s
  unsigned int transmissions; //numero de tentativas em caso de falha
  status mode; //receiver or transmitter
  struct termios oldTermios;
}linkLayer;

linkLayer link_layer;

void initDataLinkStruct(int transmissions, int timeOut, int baudRate);
void alarmHandler(int sig);
int stateMachine(unsigned char c, int state, unsigned char * msg);
int setTermios(int fd);
int llopen(int port, status mode);
int llopenTransmitter(int fd);
int llopenReceiver(int fd);
int readPacket(int fd, unsigned char *frame, int *frame_length);
bool frameSCorrect(unsigned char *response, int response_len, unsigned char C);
int llwrite(int fd,unsigned char * packet, int length, int * rejCounter);
int llread(int fd, unsigned char *packet, int *packetSize);
unsigned char *createIFrame(int *frameLength,unsigned char *packet, int packetLength);
int writePacket(int fd, unsigned char* buffer, int bufLength);
unsigned char correctBCC2(const unsigned char* buf, unsigned int size);
unsigned char *stuff(unsigned char *packet, int *packetLength);
unsigned char *destuff(unsigned char *packet, int *packetLength);
int llclose(int fd);
int llcloseTransmitter(int fd);
int llcloseReceiver(int fd);
```

linkLayer.h

A.4 linkLayer.c

```
#include "linkLayer.h"

unsigned char SET[5] = {FLAG, A_SEND, C_SET, A_SEND ^ C_SET, FLAG};
unsigned char UA[5] = {FLAG, A_SEND, C_UA, A_SEND ^ C_UA, FLAG};
unsigned char DISC[5] ={FLAG, A_SEND, C_DISC, A_SEND ^ C_DISC, FLAG};
unsigned char UA_ALT[5] = {FLAG, A_ALT, C_UA, A_ALT ^ C_UA, FLAG};
unsigned char DISC_ALT[5] ={FLAG, A_ALT, C_DISC, A_ALT ^ C_DISC, FLAG};

unsigned char RR0[5] = {FLAG, A_SEND, RR, A_SEND^RR, FLAG};
unsigned char RR1[5] = {FLAG, A_SEND, RR_ALT, A_SEND^RR_ALT, FLAG};
unsigned char REJ0[5] = {FLAG, A_SEND, REJ, A_SEND^REJ, FLAG};
unsigned char REJ1[5] = {FLAG, A_SEND, REJ_ALT, A_SEND^REJ_ALT, FLAG};

bool timeOut = false;
int count = 0;
bool ignore = false;

void initDataLinkStruct(int transmissions, int timeOut, int baudRate){

  link_layer.transmissions = transmissions;
  link_layer.timeout = timeOut;
  link_layer.baudRate = baudRate;
}

void alarmHandler(int sig){
  timeOut = true;
  count ++;
  printf("TIMED OUT\n");
}

int stateMachine(unsigned char c, int state, unsigned char * msg){
  switch (state) {

    case 0:
      if(c == msg[0])
        return 1;
    break;

    case 1:
      if(c == msg[1])
        return 2;

      if(c != msg[0])
        return 0;
    break;

    case 2:
      if(c == msg[2])
        return 3;

      if(c!= msg[0])
        return 0;
      else
        return 1;
    break;

    case 3:
      if( c == (msg[2]^msg[1]))
        return 4;

      if(c!= msg[0])
        return 0;
      if( c== msg[0])
```

```
64          return 1;
65      break;
66
67      case 4:
68        if(c != msg[0])
69          return 0;
70        else
71          return 5;
72      break;
73
74      default:
75      break;
76    }
77    return 0;
78  }
79
80  int setTermios(int fd){
81
82      struct termios oldtio, newtio;
83
84      if ( tcgetattr(fd,&oldtio) == -1) { /* save current port settings */
85          perror("tcgetattr");
86          exit(-1);
87        }
88
89        link_layer.oldTermios = oldtio; //saves old termios structure to undo change in the end
90
91        bzero(&newtio, sizeof(newtio));
92        newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
93        newtio.c_iflag = IGNPAR;
94        newtio.c_oflag = 0;
95
96        /* set input mode (non-canonical, no echo,...) */
97        newtio.c_lflag = 0;
98        newtio.c_cc[VTIME]    = 1;   /* inter-character timer unused */
99        newtio.c_cc[VMIN]     = 0;  /* blocking read until 5 chars received */
100     /*
101       VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a
102       leitura do(s) proximo(s) caracter(es)
103     */
104
105
106       tcflush(fd, TCIOFLUSH);
107
108       if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
109         perror("tcsetattr");
110         exit(-1);
111       }
112
113     return 0;
114
115   }
116
117 int llopen(int port, status mode){
118       int fd;
119       link_layer.mode = mode;
120
121       switch (port) {
122         case COM1:
123         strcpy(link_layer.port, COM1_PORT);
124         break;
125
126         case COM2:
127         strcpy(link_layer.port, COM2_PORT);
128         break;
```

```
129
130           default:
131           printf("linkLayer - llopen(): invalid port\n");
132           return -1;
133         }
134
135         fd = open(link_layer.port, O_RDWR | O_NOCTTY);
136
137         if (fd <0) {
138           perror(link_layer.port);
139           exit(-1);
140         }
141
142         if (setTermios(fd) != 0) {
143           printf("linkLayer - llopen() - setTermios: error\n");
144           return -1;
145         }
146
147         if(mode == TRANSMITTER){
148           if(llopenTransmitter(fd) <0)
149           return -1;
150         }
151
152         if(mode == RECEIVER){
153           if(llopenReceiver(fd) < 0)
154           return -1;
155         }
156
157
158         link_layer.sequenceNumber = 0;
159
160         printf("linkLayer - llopen() - SUCCESS!\n");
161
162         return fd;
163     }
164
165   int llopenTransmitter(int fd){
166     unsigned char c;
167     int state = 0;
168
169     signal(SIGALRM, alarmHandler);
170
171     do{
172
173       if(write(fd, SET, 5) != 5){
174         printf("linkLayer - llopen: error writting SET\n");
175         exit(-1);
176       }
177
178       printf("SET sent\n");
179
180       timeOut = false;
181       alarm(link_layer.timeout);
182
183       while(state != 5 && !timeOut){
184
185         if(read(fd, &c, 1) == -1){
186           printf("linkLayer - llopen: error reading\n");
187           exit(-1);
188         }
189
190           state = stateMachine(c, state, UA);
191       }
192       if(state == 5)
193               printf("UA RECEIVED\n");
```

```
194
195   } while(timeOut && count < link_layer.transmissions);
196
197   if (count == link_layer.transmissions)
198       return -1;
199     else
200       return 0;
201 }
202
203 int llopenReceiver(int fd){
204   unsigned char c;
205   int state = 0;
206
207     alarm(link_layer.timeout * link_layer.transmissions);
208
209     while(state!=5){
210      if(read(fd, &c, 1) == -1){
211        printf("linkLayer - llopen: read error\n");
212        exit(-1);
213      }
214
215      state = stateMachine(c, state, SET);
216
217    }
218    printf("SET RECEIVED\n");
219
220    if(write(fd, UA, 5) != 5){
221      printf("linkLayer - llopen: error writing UA\n");
222      exit(-1);
223    }
224
225    printf("UA SENT\n");
226
227    return 0;
228 }
229
230 int llwrite(int fd,unsigned char * packet, int length, int * rejCounter){
231   int frameLength;
232   unsigned char *frame = createIFrame(&frameLength, packet, length);
233   count = 0;
234   int state;
235
236   do{
237     if(writePacket(fd, frame, frameLength) < 0){
238       printf("llwrite: error sending packet\n");
239       return -1;
240     }
241
242     timeOut = false;
243     alarm(link_layer.timeout);
244
245     state=0;
246     unsigned char c, C;
247     while(state!=5 && !timeOut){
248       if(read(fd, &c, 1) == -1) {
249         printf("linkLayer - llopen: read error\n");
250         exit(-1);
251       }
252
253       switch (state) {
254       case 0:
255         if(c == FLAG)
256           state = 1;
257         break;
258
```

```
259         case 1:
260           if(c == A_SEND)
261             state = 2;
262           else if(c != FLAG)
263             state = 0;
264           break;
265
266         case 2:
267           if(c == RR || c == RR_ALT || c == REJ || c == REJ_ALT){
268             state = 3;
269             C = c;
270           }
271           else if(c == FLAG)
272             state = 1;
273           else
274             state = 0;
275           break;
276
277         case 3:
278           if( c == (A_SEND^C))
279             state = 4;
280           else if(c == FLAG)
281             state = 1;
282           else
283             state = 0;
284           break;
285
286         case 4:
287           if(c == FLAG)
288             state = 5;
289           else {
290             state = 0;
291           }
292           break;
293
294         default:
295           break;
296         }
297       }
298
299       if ((link_layer.sequenceNumber && C == RR) || (!link_layer.sequenceNumber && C == RR_ALT))
              {
300         printf("Recebeu RR %x, trama = %d\n", C, link_layer.sequenceNumber);
301         link_layer.sequenceNumber ^= 1;
302         alarm(0);
303       }
304       else if ((link_layer.sequenceNumber && C == REJ) || (!link_layer.sequenceNumber && C ==
              REJ_ALT)){
305         printf("Recebeu REJ %x, trama = %d\n", C, link_layer.sequenceNumber);
306         timeOut = true;
307         alarm(0);
308       }
309     }while(timeOut && count < link_layer.transmissions);
310
311     if (count == link_layer.transmissions)
312       return -1;
313
314     return 0;
315   }
316
317   int llread(int fd, unsigned char *packet, int *packetSize){
318
319     unsigned char c;
320     int state=0;
321     int res;
```

```
322
323       alarm(5);
324
325     while(state!=5){
326       if((res = read(fd, &c, 1)) == -1){
327         printf("linkLayer - llopen: read error\n");
328         return -1;
329       }
330       else if(res == 0 && state == 4)
331         break;
332
333       switch (state) {
334         case 0:
335           if(c == FLAG)
336             state = 1;
337           break;
338
339         case 1:
340           if(c == A_SEND)
341             state = 2;
342           else if(c != FLAG)
343             state = 0;
344           break;
345
346         case 2:
347           if(c == (link_layer.sequenceNumber << 6))
348             state = 3;
349           else if(c == FLAG)
350             state = 1;
351           else
352             state = 0;
353           break;
354
355         case 3:
356           if( c == (A_SEND^(link_layer.sequenceNumber << 6)))
357             state = 4;
358           else if(c == FLAG)
359             state = 1;
360           else
361             state = 0;
362           break;
363
364         case 4:
365           if(c == FLAG)
366             state = 5;
367           else {
368             packet[(*packetSize)++] = c;
369           }
370           break;
371
372         default:
373           break;
374       }
375     }
376
377     unsigned char* destuffed;
378     destuffed = destuff(packet, packetSize);
379     memcpy(packet, destuffed, *packetSize);
380
381     if (correctBCC2(destuffed, *packetSize) == 0){
382       ignore = false;
383       if (link_layer.sequenceNumber)
384         write(fd, RR0, 5);
385       else
386         write(fd, RR1, 5);
```

```
387
388       printf("Enviou RR, sequenceNumber:%d\n", link_layer.sequenceNumber);
389
390       link_layer.sequenceNumber ^= 1;
391     }
392     else {
393       ignore = true;
394
395       if (link_layer.sequenceNumber)
396         write(fd, REJ0, 5);
397       else
398         write(fd, REJ1, 5);
399
400       printf("Enviou REJ, sequenceNumber:%d\n", link_layer.sequenceNumber);
401     }
402     return ignore;
403   }
404
405   unsigned char *createIFrame(int *frameLength,unsigned char *packet, int packetLength){
406     unsigned char *stuffPacket = stuff(packet, &packetLength);
407     *frameLength = packetLength + 5; //packetLength + 5 flags
408     unsigned char *frame = (unsigned char *)malloc(*frameLength * sizeof(unsigned char));
409
410     frame[0] = FLAG;
411     frame[1] = A_SEND;
412     frame[2] = link_layer.sequenceNumber << 6;
413     frame[3] = frame[1] ^ frame[2];
414
415     memcpy(frame + 4, stuffPacket, packetLength); //copies stuffed packet to frame
416     frame[*frameLength-1] = FLAG;
417
418     return frame;
419   }
420
421   int writePacket(int fd, unsigned char* buffer, int bufLength){
422     int chars;
423
424     chars = write(fd, buffer, bufLength);
425
426     if(chars < 0) {
427       printf("error writing\n");
428       return -1;
429     }
430     else
431       printf("Enviou I com %d bytes\n", chars);
432
433     return 0;
434   }
435
436   unsigned char correctBCC2(const unsigned char* buf, unsigned int size) {
437       unsigned char BCC2 = buf[0];
438
439       unsigned int i;
440       for (i = 1; i < size; ++i)
441           BCC2 ^= buf[i];
442
443       return BCC2;
444   }
445
446   unsigned char *stuff(unsigned char *packet, int *packetLength){
447
448     unsigned char* stuffed = (unsigned char *)malloc(2 * (*packetLength));
449
450     unsigned char BCC2 = 0;
451     int i = 0, j = 0;
```

```
452
453     //Calcular o BCC2
454     for(i = 0; i < *packetLength; i++){
455       BCC2 ^= packet[i];
456     }
457
458     packet[*packetLength] = BCC2;
459     *packetLength = *packetLength + 1;
460
461     //Fazer stuff
462     for(i = 0; i < *packetLength; i++){
463
464        if(packet[i] == FLAG || packet[i] == ESC){
465         stuffed[j] = ESC;
466         stuffed[++j] = packet[i] ^ BYTE_STUFF;
467         j++;
468       }
469
470       else{
471         stuffed[j] = packet[i];
472         j++;
473       }
474     }
475
476     *packetLength = j;
477
478     return stuffed;
479
480   }
481
482   unsigned char *destuff(unsigned char *packet, int *packetLength){
483
484     unsigned char* destuffed = (unsigned char*)malloc(((*packetLength)));
485
486     int i = 0, j = 0;
487
488     for (; i < *packetLength; i++){
489       if(packet[i] == ESC){
490         destuffed[j] = packet[i + 1] ^ BYTE_STUFF;
491         i++;
492       } else
493         destuffed[j] = packet[i];
494
495         j++;
496     }
497
498     *packetLength = j;
499
500     return destuffed;
501   }
502
503   int llclose(int fd){
504     if(link_layer.mode == TRANSMITTER)
505     if(llcloseTransmitter(fd) <0)
506     return -1;
507
508     if(link_layer.mode == RECEIVER)
509     if(llcloseReceiver(fd) < 0)
510     return -1;
511
512     if ( tcsetattr(fd, TCSANOW, &link_layer.oldTermios) == -1) {
513       perror("tcsetattr");
514       exit(-1);
515     }
516
```

```
517     close(fd);
518     return 0;
519   }
520
521   int llcloseTransmitter(int fd){
522     unsigned char c;
523     int state = 0;
524
525     do{
526
527       if(write(fd, DISC, 5) != 5){
528         printf("linkLayer - llclose: error writting DISC\n");
529         return -1;
530       }
531       printf("DISC SENT!\n");
532
533       timeOut = false;
534       alarm(link_layer.timeout);
535
536       while(state != 5 && !timeOut){
537
538         if(read(fd, &c, 1) == -1){
539           printf("linkLayer - llclose: error reading\n");
540           return -1;
541         }
542           state = stateMachine(c, state, DISC_ALT);
543       }
544       printf("RECEIVED DISC\n");
545
546     } while(timeOut && count < link_layer.transmissions);
547
548     if(write(fd, UA_ALT, 5) != 5){
549       printf("linkLayer - llclose: error writting UA\n");
550     }
551
552     printf("UA SENT\n");
553
554
555     return 0;
556   }
557
558   int llcloseReceiver(int fd){
559     unsigned char c;
560     unsigned char d;
561     int state = 0;
562
563       alarm(link_layer.timeout * link_layer.transmissions);
564
565     while(state != 5){
566
567        if(read(fd, &c, 1) == -1){
568          printf("linkLayer - llclose: error reading DISC\n");
569          return -1;
570        }
571
572        state = stateMachine(c, state, DISC);
573
574      }
575
576      printf("DISC RECEIVED!\n");
577
578      if(write(fd, DISC_ALT, 5) != 5){
579        printf("linkLayer - llclose: error writing DISC\n");
580        exit(-1);
581      }
```

```
582
583     printf("DISC SENT!\n");
584
585     state = 0;
586
587     while(state != 5) {
588       if(read(fd, &d, 1) == -1){
589         printf("linkLayer - llclose: error reading UA\n");
590         return -1;
591       }
592       state = stateMachine(d, state, UA_ALT);
593    }
594      printf("UA RECEIVED\n");
595
596     return 0;
597 }
```

linkLayer.c

A.5 utilities.h

```
#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define BAUDRATE B38400
#define TIMEOUT 3
#define TRANSMISSIONS 3

#define COM1 0
#define COM2 1
#define COM1_PORT "/dev/ttyS0"
#define COM2_PORT "/dev/ttyS1"

#define FALSE 0
#define TRUE 1

#define ESC 0x7D
#define BYTE_STUFF 0x20
#define HEADER_SIZE 6

#define FILE_SIZE 0
#define FILE_NAME 1

#define DATA 1
#define START 2
#define END 3

#define FLAG 0x7E

#define A_SEND 0x03
#define A_ALT 0x01

#define C_SET 0x03
#define C_UA 0x07
#define C_DISC 0x0B

#define SET_BCC 0x00

#define RR 0x05
#define REJ 0x01
#define RR_ALT 0x85
#define REJ_ALT 0x81

#define WAIT 4000

#define PACKET_SIZE 256
#define PACKET_HEADER_SIZE 4
#define DATA_PACKET_SIZE PACKET_SIZE - PACKET_HEADER_SIZE

#define SFRAMELEN 5

typedef enum {TRANSMITTER, RECEIVER} status;
```

utilities.h

A.6 makefile

```
gs: applicationLayer.c linkLayer.c
	gcc -Wall -o t1 applicationLayer.c linkLayer.c

clean:
	rm -f t1
```

makefile