



1º Trabalho Laboratorial – Data Link Protocol

Relatório

Redes de computadores

Margarida Mesquita Leal – up201906884

Miguel Augusto Marombal Araújo - up201905077

Sumário

O presente relatório visa sintetizar e explicar o primeiro trabalho prático-laboratorial de Redes de Computadores, cujo propósito foi o desenvolvimento e implementação de um protocolo de ligação de dados capaz de estabelecer a ligação entre dois computadores (através de uma porta série) e transferir, de forma assíncrona, informação entre os mesmos.

O protocolo desenvolvido cumpriu os requisitos necessários e transferiu o ficheiro desejado de forma eficaz quer nas condições ideias quer quando foi inserido ruído no sistema ou é interrompida temporariamente a ligação.

Introdução

De acordo com o guião fornecido, foi implementado um protocolo de ligação de dados o qual foi, durante o seu desenvolvimento e até à reta final, testado de diversas formas. O guião incidirá na explicação teórica de conceitos essenciais ao desenvolvimento desta aplicação. Está organizado da seguinte forma:

- **Arquitetura e Estrutura do código** - blocos funcionais e interfaces (Arquitetura) APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura (Estrutura do código).
- **Protocolo de ligação lógica** - identificação dos principais aspectos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código.
- **Validação** - descrição dos testes efetuados.
- **Eficiência do protocolo de ligação de dados** - caracterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.
- **Conclusão** - síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.
- **Anexo I** - código fonte

Arquitetura e estrutura do código

Por questões de organização o código foi dividido num total de 5 ficheiros **linklayer.c**, **geral.c**, **receiver.c**, **transmitter.c** e **defines.h** (a seguir enunciados e descritos criteriosamente). Esta estrutura permitiu uma organização do código de modo que seja intuitiva a sua leitura e análise.

Linklayer.c

O ficheiro **Linklayer.c** é o ficheiro principal onde estão implementadas as funções do protocolo de ligação de dados, esta camada é responsável pela conexão entre os computadores e o envio da informação entre os mesmos. Este ficheiro serve-se das funções implementadas nos ficheiros auxiliares de modo a garantir a sua legibilidade e organização. Conta deste modo com as funções `llopen`, `llwrite`, `llread` e `llclose` cuja função é descrita a seguir:

- `llopen` – Responsável por configurar a porta série e garantir o estabelecimento da ligação segundo o protocolo (Envio da trama SET por parte do Transmissor e resposta com UA da parte do receptor). Deve também guardar informações úteis às restantes funções do ficheiro como o modo em que opera o tempo de timeout em segundos e o número de retransmissões. Retorna “1” em caso de sucesso, “-1” em perante um eventual erro.
- `llwrite` – Responsável por, após aberta a conexão, preparar a trama para ser enviado, isto inclui a inserção do cabeçalho e da terminação segundo o protocolo, bem como a realização de stuffing (cuja função é explicada de seguida). Após o envio da trama o `llwrite` deve também esperar uma resposta do receptor e atuar segundo ela, isto é, caso a mesma não chegue no tempo definido deve ser considerada perdida e reenviada até um total de um número definido de vezes (tanto o tempo como o número de retransmissões são configuradas no `llopen`). Retorna o número de bytes enviados em caso de sucesso, “-1” em perante um eventual erro.
- `llread` – Responsável pela leitura da informação que chega através da porta série. Deve verificar a integridade da mensagem e rejeitar pacotes corrompidos, deve também saber identificar pacotes repetidos, descartá-los reenviando de qualquer forma a confirmação que tal pacote já foi recebido. E perante um pacote recebido com sucesso deve confirmar a sua receção bem como proceder à remoção dos cabeçalhos e terminações que não representam dados úteis à camada de aplicação e realização do destuffing. Retorna o número de bytes enviados em caso de sucesso, “-1” em perante um eventual erro.

- llclose – Responsável por comunicar o fecho da ligação segundo o protocolo (envio do DISC pelo transmissor, resposta com um DISC por parte do receptor e, finalmente, resposta com UA pelo transmissor). No fim, deve fechar a ligação configurada inicialmente no llopen. Retorna “1” em caso de sucesso, “-1” em perante um eventual erro.

Geral.c

O ficheiro em **Geral.c** serve como auxiliar de modo a conter as funções que mais vezes serão repetidas durante o código. Sendo as mesmas:

- setupSerialTerminal – Responsável por configurar a porta série para comunicação de dados.
- closeSerialTerminal – Responsável por fechar a comunicação porta série.
- setupFrameFormat – Responsável pela adição de cabeçalho e terminação ao payload respeitando as regras definidas pelo protocolo.
- resetFrameFormat – Responsável por remover o cabeçalho e terminação.
- stuffing – Responsável pela remoção de FLAGs e ESCAPEs no payload, aquando de uma FLAG a mesma deve ser trocada pelo ESCAPE (0x7D, neste protocolo) e o byte seguinte deve ser o resultado do XOR da FLAG com 0x20 cujo resultado deve ser 0x5E, de forma análoga quando é encontrado um ESCAPE no payload este também deve ser sinalizado colocando no byte seguinte o XOR do ESCAPE com 0x20. Desta forma é garantida a transparência do código e que o cabeçalho e a terminação são entendidos como tal visto serem os únicos onde FLAGs podem aparecer.
- destuffing – Responsável pelo processo oposto ao stuffing recuperando a mensagem original.
- convertBaundRate – Responsável pela conversão do BaundRate de “int” para “speed_t”
- calculaBCC – Responsável pelo cálculo do BCC (Independet Protection Field) necessário para a identificação de erros.
- Statistics – Responsável por imprimir no terminal as estatísticas recolhidas da transmissão de dados.
- resetStats – Responsável por colocar a 0 todos os campos da estrutura que guarda as estatísticas.

- toggleNS – Responsável por alterar o Ns entre os dois valores possíveis (0x00 ou 0x02) este valor representará o campo Control Field do cabeçalho de tramas, RR e REJs e permite com o protocolo STOP and WAIT identificar pacotes repetidos no caso do recetor bem como identificar que pacote o mesmo está pronto para receber ou que pacote está a rejeitar.

Receiver.c

O ficheiro **Receiver.c** implementa a função responsável pela abertura da ligação no modo de recetor.

- Llopen_receiver – Responsável por abrir a conexão como recetor.

Transmitemer.c

O ficheiro **Transmitemer.c** implementa a função responsável pela abertura da ligação no modo de transmissor.

- Llopen_transmitter – Responsável por abrir conexão como transmissor.

Defines.h

O ficheiro **Defines.h** é responsável pela definição de variáveis e da estrutura usada para as recolher dados estatísticos acerca da ligação.

Casos de uso

O caso de uso principal da aplicação é a interface com o utilizador onde este indica como primeiro argumento a porta série (no caso do laboratório pode ser por exemplo: /dev/ttyS0) o modo em que pretende operar sendo “rx” para recetor e “tx” para transmissor e ainda o caminho para o ficheiro a enviar/guardar (por exemplo, “penguin.gif”).

Após serem processados os argumentos são executadas as seguintes tarefas:

- Configuração e estabelecimento de ligação entre os dois computadores através da porta série.
- Transmissão dos dados do ficheiro com o nome indicado no terceiro parâmetro respeitando o protocolo Stop and Wait.
- Receção dos dados, análise e envio das respetivas confirmações (RR) ou rejeições (REJ).

- Do lado do recetor os dados são guardados no ficheiro com o nome indicado no terceiro parâmetro.
- Terminação e fecho da ligação.

Protocolo de ligação lógica

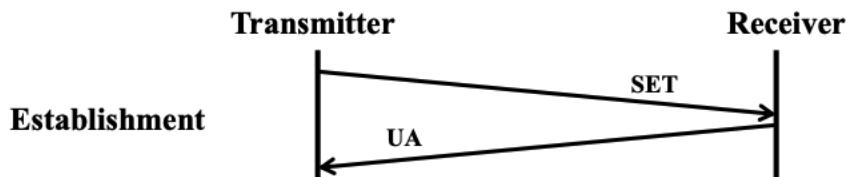
O protocolo de ligação logica é responsável por fornecer as funções necessárias para a comunicação de dados entre os dois computadores ligados através da porta serie. O protocolo tem como principais objetivos:

- **Configurar a porta série;**
- **Estabelecer a ligação entre os dispositivos-** no nosso caso estabelece ligação entre os dois pontos (dois computadores, ligados de ponto a ponto através de uma porta VGA);
- **Transferir os dados através da porta serie, garantindo que a realização de stuffing/destuffing;**
- **Deteção de erros quer por introdução de ruido quer pelo interromper da ligação aquando da transferência de dados;**

Sendo assim, são responsáveis pela implementação dos requisitos acima mencionados as seguintes funções:

1. **`int llopen(linkLayer connectionParameters);`**

A função llopen tem como objetivo garantir o estabelecimento da ligação entre o emissor e o recetor, devendo, deste modo, retornar “1” quando tal estabelecimento é realizado com sucesso ou “-1” quando ocorrer algum erro que comprometa o estabelecimento da ligação. Para tal segue a seguinte logica definida pelo protocolo (ilustrada na figura seguinte).



Deve ser enviado pelo transmissor uma trama SET, esta trama tem uma sequência específica definida pelo protocolo. Quando o recetor receber o SET deve confirmar que o recebeu enviando de volta uma trama específica designada por UA.

Se durante este processo alguma trama se perder ou for corrompida o transmissor deve proceder ao reenvio do SET, lembrando que o tempo que o transmissor espera até considerar que o

pacote foi perdido bem como o número de retransmissões a fazer antes de considerar que foi impossível abrir ligação são parâmetros definidos pela camada de aplicação e passados à função llopen através da estrutura que serve como único parâmetro da função “linklayer connectionParameters”.

Para além desta informação, a estrutura fornece também informação acerca do modo em que o llopen está a ser chamado, isto é, ou receiver ou transmitter bem como o serialPort necessário. Assim sendo, o llopen começa por identificar o modo em que deve atuar (Receiver ou Transmitter), guardar em variáveis globais informações que serão úteis no decorrer do código (número de tentativas de retransmissão, tempo de time out em segundos e modo de operação) e posteriormente chamar as funções llopen_receiver ou llopen_transmitter conforme o modo em que foi chamado. Estas mesmas funções estão implementadas nos ficheiros receiver.c e transmitter.c por uma questão de organização de código.

2. `int llwrite(char* buf, int bufSize);`

A função llwrite tem como responsabilidade permitir ao emissor enviar os dados através da porta série. Com a ligação já estabelecida e configurada pela função llopen descrita anteriormente.

A função llwrite tem como parâmetros o apontador para o buf que pretende transmitir bem como um inteiro que indica o tamanho do mesmo. O seu retorno é “-1” caso algum erro aconteça no processo (quer seja este um erro nos parâmetros quer um erro na transmissão de dados, por exemplo, se se interromper a ligação física entre os dois computadores), caso não ocorra nenhum erro a função retorna o número de bytes enviados.

Para garantir um envio interpretável dos dados bem como a necessária recuperação de erros, a função llwrite é responsável pela introdução do cabeçalho e finalização da trama. Sendo o cabeçalho composto por FLAG, Address field, Control field e BCC1 (Independet protection field). Já a terminação é composta pelo BCC2 e FLAG. Esta estrutura permitirá ao Recetor reconhecer o início da trama, o seu fim, perceber através do Control Field se está a receber o pacote desejado ou um repetido e (através dos campos de proteção BCC1 e BCC2) identificar erros na transmissão.

É ainda necessário e da responsabilidade do llwrite fazer o stuffing da trama, de modo a garantir que não existem FLAGS nem ESCAPES na mensagem além dos necessários para definir o início e fim da trama. Para implementar estas funcionalidades foram usadas as funções setupFrameFormar, stuffing e calculaBCC, todas implementadas no ficheiro geral.c para uma melhor organização do código.

Esta função deve também garantir as necessárias retransmissões caso receber como resposta do Recetor mensagem de REJ ou caso, após “timeout” segundos não tenha recebido nenhuma confirmação de receção do pacote.

3. `int llread(char* packet);`

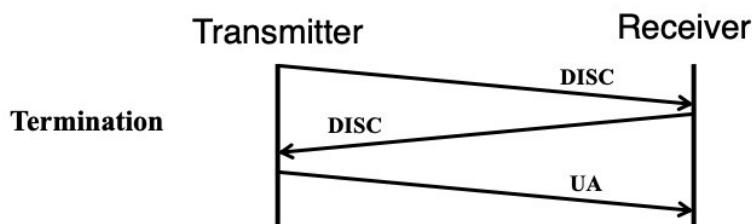
A função llread tem como objetivo permitir ao recetor receber através da porta serie a trama enviada pelo transmissor e guardá-la no apontador que serve como único parâmetro da função, packet. O retorno deve ser “-1” quando algum erro acontecer no processo ou o número de bytes recebidos que fazem parte do payload da mensagem.

Para cumprir estes requisitos a função llread deve ser capaz de identificar o inicio e o fim da mensagem através das FLAGS, verificar (através do Control Field se está a receber o pacote pretendido ou uma retransmissão de um pacote repetido, neste caso deve-o descartar e reconfirmar que já o recebeu), deve também verificar se os BCCs (quer o BCC1 responsável pelo controlo de erros no Header, quer o BCC2 responsável pelo controlo de erros no payload) estão corretos e caso não estejam enviar um REJ indicando ao transmissor a rejeição do pacote em questão para que tal seja retransmitido (respeitando o protocolo STOP and WAIT).

Perante um pacote que respeita todos estes requisitos o llread deve enviar uma confirmação que está pronto a receber o próximo pacote ao transmissor e ser capaz de remover os cabeçalhos e finalizações do pacote visto que não representam os dados do payload e fazer o respetivo destuffing para recuperar a trama original. Para cumprir com isto são usadas as funções resetFrameFormat, destuffing e calculaBCC presentes no ficheiro geral.c.

```
4. int llclose(int showStatistics);
```

A função llclose tem como finalidade terminar a ligação entre o emissor e o receptor. Para isso deve a seguinte logica (representada na figura seguinte).



O emissor deve enviar uma trama DISC, o receptor quando receber a mesma deve enviar também um DISC (de realçar que os DISCs enviados pelo emissor e receptor se distinguem pelo seu address field e consequentemente pelo seu BCC), por fim deve o transmissor confirmar que recebeu o DISC do emissor enviando um UA. Também para este caso o Transmitter deve esperar timeout segundos pela resposta e caso não a obtenha deve retransmitir o DISC o número pré-definido de vezes.

Para cumprir o protocolo acima descrito esta função deve começar por identificar em que modo opera “Transmitter” ou “Receiver” e cumprir o protocolo segundo as funções que lhe competem.

Existe também um parâmetro ShowStatistics que quando igual a TRUE deve imprimir no terminal um conjunto de estatísticas acerca da ligação recolhidas durante o envio/recepção de dados.

Validação

Durante o desenvolvimento e no fim do trabalho foi necessário, de forma a confirmar se todas as funções cumpriam com os requisitos pretendidos, realizar testes de modo a pôr à prova a resiliência da implementação aos diferentes cenários possíveis. Para tal foram realizados testes com:

- Textos de várias dimensões
- Diferentes imagens também diferentes dimensões.
- Introdução de erros na mensagem (simulando a possível existência de ruído na ligação)
- Interrupção temporária da ligação (quer através de software com a utilização da função sleep() quer interrompendo fisicamente a ligação)

Os testes descritos permitiram testar a robustez da implementação perante cenários em que o ruído existente é capaz de adulterar a informação que pretendemos transmitir, em que pacotes são perdidos e onde ambos os problemas acontecem em simultâneo.

Mudar de ambiente foi um teste crucial. A existência de diferenças entre portas virtuais e a porta física do laboratório traz variáveis e erros do quais não estamos a contar. Depois de se testar o código no laboratório tiveram de se corrigir alguns pormenores, tais como, time out e tempo de escrita e leitura.

Conclusões

O desenvolvimento do código e da aplicação permitiu uma aprendizagem aprofundada dos conceitos estudados: protocolo de ligação de dados, protocolo “Stop and Wait” e cálculo do BCC para identificação de erros, assim como o desenvolvimento um projeto com múltiplos ficheiros e alguma extensão de código.

Foi também fundamental perceber a importância da independência entre camadas. Visto que o Protocolo de Ligação Lógica pode ser completamente usado com recurso às quatro funções (llopen, llwrite, llread, llclose) sem o conhecimento detalhado de todas as funções e ficheiros que estão por detrás da sua implementação.

Embora com alguns desafios, o trabalho foi realizado com sucesso e cumpriu todos os objetivos pretendidos. Foi desenvolvido um protocolo de ligação de dados capaz de enviar, com sucesso informação entre dois computadores distintos, bem como identificar e recuperar de eventuais erros durante a transmissão.

Anexo I

Defines.h

```
1 #ifndef DEFINES_H
2 #define DEFINES_H
3
4
5 #define FLAG 0x7e
6 #define A_1 0x03 // A_tx
7 #define A_2 0x01 // A_rx
8
9 #define BCC_1 (A_1^C)
10 #define BCC_2 (A_2^C)
11
12 #define BCC_DISC_tx (A_1^C_DISC)
13 #define BCC_DISC_rx (A_2^C_DISC)
14
15 #define C 0x03
16 #define C_SET 0x03
17 #define C_DISC 0x0b
18 #define C_UA 0x07
19
20 #define C_random 0x04
21 #define C_random1 0x05
22 #define C_S0 0x00
23 #define C_S1 0x02
24 #define C_RR0 0x01
25 #define C_RR1 0x21
26 #define C_REJ0 0x05
27 #define C_REJ1 0x25
28
29 #define ESC 0x7D
30
31
32 #define XOR 0x20
33
34 #define STATE0 0 // Start
35 #define STATE1 1 // Flag RCV
36 #define STATE2 2 // A RCV
37 #define STATE3 3 // C RCV
38 #define STATE4 4 // BCC OK
39 #define STATE5 5 // STOP
40 // para maquinas mais avançadas
41 #define STATE6 6
42 #define STATE7 7
43 #define STATE8 8
44
45
46 typedef struct stats{
47     int num_frames;
48     int num_bytes;
49     int num_databytes;
50     int num_timeouts;
51     int num_REJ;
52 } stats;
53
54 #endif
```

Geral.c

```

1  #include "geral.h"
2
3
4  /*
5   *Function: Setup the communication
6   *Returns: returns fd (file descriptor) on sucess, -1 on error
7   */
8  int setupSerialTerminal(linkLayer connectionParameters){
9      int fd = open(connectionParameters.serialPort, _O_RDWR | _O_NOCTTY );
10     if (fd <0) {perror(connectionParameters.serialPort); exit(-1);}
11
12     if ( tcgetattr(fd,&oldtio) == -1) { /* save current port settings */
13         perror("tcgetattr");
14         exit(-1);
15     }
16
17     bzero(&newtio, sizeof(newtio));
18     newtio.c_iflag = IGNPAR;
19     newtio.c_oflag = 0;
20
21     /* set input mode (non-canonical, no echo,...) */
22     newtio.c_lflag = 0;
23
24
25     newtio.c_cc[VTIME] = (connectionParameters.timeOut * 10); /* inter-character timer unused */
26     newtio.c_cc[VMIN] = 0; /* blocking read until 5 chars received */
27
28     tcflush(fd, TCIOFLUSH);
29
30     if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
31         perror("tcsetattr");
32         exit(-1);
33     }
34     return fd;
35 }
36
37 /*
38 *Function: Closes the communication
39 *Returns: returns 1 on sucess, -1 on error
40 */
41 int closeSerialTerminal(int fd){
42 //tcflush(fd, TCIOFLUSH);
43     if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
44         perror("tcsetattr");
45         exit(-1);
46     }
47     sleep(1);
48     close(fd);
49     return 1;
50 }
51
52
53 /*
54 *Function:
55 *Returns:
56 */
57 int setupFrameFormat(char* buf, char* frame, int bufferSize){
58     if(!buf){if(bufferSize<0) return -1;
59     char BCC2 = calculateBCC(buf, bufferSize);
60
61     frame[0] = FLAG;
62     frame[1] = A_;
63     frame[2] = C;
64     frame[3] = BCC_1; // MALMAMALASDIAHSUOIAHDUHADUWHUIDHAUDHUAIH
65     for(int i = 0; i < bufferSize; i++){
66         frame[i+4] = buf[i];

```

```

67     }
68     frame[bufSize + 4] = BCC2;
69     frame[bufSize + 5] = FLAG;
70     //imprime(frame, size+6);
71     return bufSize+6; /// MUITO CUIDADO
72 }
73 */
74 */
75 */
76 */
77 *Function:
78 *Return:
79 */
80 int resetFrameFormat(char* buf, int bufSize){
81     if((!buf)||!(bufSize<0)) return -1;
82     /* Tirar a FLAG, A, C, BCC do inicio do buf */
83     for(int i = 0; i < bufSize - 2; i++){
84         buf[i] = buf[i+4];
85     }
86     return bufSize - 6;
87 }
88 */
89 */
90 */
91 *Function: Stuff the buf
92 *Return: The size of the new buf on sucess, -1 on error
93 */
94 int stuffing(char* buf, int bufSize){
95     if((!buf)||!(bufSize<0)) return -1;
96     char aux1, aux2;
97     for(int i = 3; i < bufSize-1; i++){
98         if(buf[i] == ESC){
99             bufSize++;
100            aux1 = buf[i+1];
101            aux1 = buf[i+1];
102            buf[i+1] = (ESC^XOR); //this should be 0x5e
103            for(int j = i+2; j <= bufSize ;j++){
104                aux2 = buf[j];
105                aux1 = aux1 ^ aux2;
106                buf[j]=aux1;
107                aux1 = aux2;
108            }
109        }
110    }
111    for(int i = 3; i<bufSize-1; i++){
112        if(buf[i] == FLAG){
113            if(buf[i]==FLAG){
114                buf[i]=ESC;
115                bufSize++;
116                aux1=buf[i+1];
117                buf[i+1]=(FLAG^XOR); //this should be
118                for(int j = i+2; j <= bufSize; j++){
119                    aux2 = buf[j];
120                    buf[j]=aux1;
121                    aux1=aux2;
122                }
123            }
124        }
125    }
126    return bufSize;
127 }
128 */
129 *Function: Destuffs the buf
130 *Return: The size of the new buf on sucess, -1 on error
131 */
132 int destuffing(char* buf, int bufSize){
133     if((!buf)||!(bufSize<0)) return -1;
134     for(int i = 0; i < bufSize; i++){
135         if(buf[i] == ESC){
136             if(buf[i+1] == (FLAG^XOR)){
137                 for(int j = i; j < bufSize; j++){
138                     buf[j] = buf[j+1];
139                 }
140                 bufSize--;
141                 buf[i] = FLAG;
142             }
143             else if(buf[i+1] == (ESC^XOR)){
144                 for(int j = i; j < bufSize; j++){
145                     buf[j] = buf[j+1];
146                 }
147                 bufSize--;
148                 buf[i] = ESC;
149             }
150             else{
151                 return -1;
152             }
153         }
154     }
155     return bufSize;
156 }
157 */
158 speed_t convertBaudRate(int baud){
159     switch (baud) {
160     case 9600:
161         return B9600;
162     case 19200:
163         return B19200;
164     case 38400:
165         return B38400;
166     case 57600:
167         return B57600;
168     case 115200:
169         return B115200;
170     case 230400:
171         return B115200;
172     case 460800:
173         return B230400;
174     case 500000:
175         return B500000;
176     case 576000:
177         return B576000;
178     case 921600:
179         return B921600;
180     case 1000000:
181         return B1000000;
182     case 1152000:
183         return B1152000;
184     case 1500000:
185         return B1500000;
186     case 2000000:
187         return B2000000;
188     case 2500000:
189         return B2500000;
190     case 3000000:
191         return B3000000;
192     case 3500000:
193         return B3500000;
194     case 4000000:
195         return B4000000;
196     default:
197         return B4000000;
198     }
199 }
200 */
201 */
202 char calculaBCC(char* frame, int frameSize){
203     char BCC2 = 0x00;
204     for(int k = 0; k < frameSize; k++){
205         BCC2 ^= frame[k];
206     }
207     return BCC2;
208 }
209 */
210 void Statistics(stats stats_){
211     printf("\n");
212     printf("*****\n");
213     printf("*****\n");
214     printf("*****\n");
215     printf("*****\n");
216     printf("*****\n");
217     //STATS
218     printf("Number of received/transmited frames: %d\n", stats_.num_frames);
219     printf("Number of received/transmited BYTES -> Total: %d (%d of data/%d of overhead)\n", stats_.num_bytes, stats_.num_databytes, stats_.num_bytes-stats_.num_databytes);
220     printf("Number of REJ: %d\n", stats_.num_REJ);
221     printf("Number of timeouts: %d\n", stats_.num_timeouts);
222     //REJ
223     printf("\n");
224     return;
225 }
226 */
227 */
228 void resetStats(stats stats_){
229     stats_.num_REJ = 0;
230     stats_.num_timeouts = 0;
231     stats_.num_frames = 0;
232     stats_.num_bytes = 0;
233     stats_.num_databytes = 0;
234 }
235 */
236 char toggleNs(char Ns){
237     if(Ns == C_S0) return C_S1;
238     else return C_S0;
239 }
240 */
241 */
242 /* Funcao: Imprimir no formato hexadecimal o conteudo do parametro "buf"
243 */
244 void imprime(char* buf, int bufSize{
245     if((!buf)||!(bufSize<0)) return;
246     for(int i = 0; i < bufSize; i++){
247         printf("%#02x ", buf[i]);
248     }
249     printf("\n");
250 }

```

Geral.h

```

1 #ifndef GERAL_H
2 #define GERAL_H
3
4 #include "linklayer.h"
5 #include "defines.h"
6
7 static struct termios oldtio, newtio;
8
9
10 int setupSerialTerminal(linkLayer connectionParameters);
11 int closeSerialTerminal(int fd);
12 int setupFrameFormat(char* buf, char* frame, int bufSize);
13 int resetFrameFormat(char* buf, int bufSize);
14 int stuffing(char* buf, int bufSize);
15 int destuffing(char* buf, int bufSize);
16 speed_t convertBaudRate(int baud);
17 char calculaBCC(char* frame, int frameSize);
18 void Statistics(stats tats_);
19 void resetStats(stats stats_);
20 char toggleNs(char Ns);
21
22 void imprime(char* buf, int bufSize); // for debug
23
24
25 #endif

```

LinkLayer.c

```

1 /* INCLUDES .h*/
2 #include "linklayer.h"
3 #include "receiver.h"
4 #include "transmitter.h"
5 #include "geral.h"
6 #include "defines.h"
7
8 static struct termios oldtio, newtio;
9 static int fd;
10 static int numTries_Receiver, timeOut_Receiver;
11 static int numTries_Transmitter, timeOut_Transmitter;
12 static int TYPE; // TYPE = 0 (TX) | TYPE = 1 (RX)
13 int timeouts = 0, flag_ = 1, conta = 0;
14 int func; // func = 0 on llopen; = 1 on llwrite,llread; = 2 on llclose
15 char Ns = C_S0;
16
17 static int x1 = 2; // for deb proposes
18
19 static struct stats stats_;
20
21 void timeout_()
22 {
23     printf("timeout: %d..\n", (timeouts+1));
24     flag_ = 1;
25     timeouts++;
26     if(conta) stats_.num_timeouts++;
27 }
28
29 // Opens a connection using the "port" parameters defined in struct linkLayer, returns "-1" on error and "1" on sucess
30 int llopen(linkLayer connectionParameters){
31     /* Reiniclar timeouts */
32     timeouts = 0;
33     conta = 0; // PARA AS ESTATISTICAS
34     fd = setupSerialTerminal(connectionParameters);
35     if(connectionParameters.role == RECEIVER){
36         //... abre como receiver
37         resetStats(stats_);
38         TYPE = connectionParameters.role;
39         numTries_Receiver = connectionParameters.numTries;
40         timeOut_Receiver = connectionParameters.timeOut;
41         return llopen_receiver(connectionParameters, fd);
42     }
43     if(connectionParameters.role == TRANSMITTER){
44         //... abre como transmitter
45         resetStats(stats_);
46         TYPE = connectionParameters.role;
47         numTries_Transmitter = connectionParameters.numTries;
48         timeOut_Transmitter = connectionParameters.timeOut;
49         return llopen_transmitter(connectionParameters, fd);
50     }
51
52     return -1;
53 }
54
55 /* Sends data in buf with size bufferSize
56 * Function:
57 * Return: "-1" on error and "1" on sucess
58 */
59 int llwrite(char* buf, int bufferSize){
60     // controla dos parametros
61     if(!buf || (bufSize<0) || (bufSize>MAX_PAYLOAD_SIZE)) return -1;
62
63     conta = 1; // PARA AS ESTATISTICAS
64     //sleep(1);
65     /* Reiniclar timeouts */
66     timeouts = 0, flag_ = 1;

```

```

67
68     char frame[7+2+MAX_PAYLOAD_SIZE*2]; /*frame auxiliar*/
69     int error = 0;
70     int STOP = FALSE, count = 0, i = 0, flag_encontrada = 0, bufSize_aux, STATE = 0;
71     char answer[5], AUX_1;
72     char REJ0[] = {FLAG, A_1, C_REJ0, (A_1^C_REJ0), FLAG};
73     char REJ1[] = {FLAG, A_1, C_REJ1, (A_1^C_REJ1), FLAG};
74     char RR0[] = {FLAG, A_1, C_RR0, (A_1^C_RR0), FLAG}; //MAL OS COISOS
75     char RR1[] = {FLAG, A_1, C_RR1, (A_1^C_RR1), FLAG};
76     /* Controle */
77     //printf("llwrite here... im will setup this: %s\n", buf);
78     //imprime(buf, bufSize);
79
80     /*PARTE 1 - PREPARAR O FRAME***** */
81     /* FrameFormat */
82     stats_.num_databytes += bufSize;
83
84     int new_bufSize = setupFrameFormat(buf, frame, bufSize);
85     if(new_bufSize == -1) return -1;
86     /* Stuffing */
87     /* CORRECAO DO C*/
88     frame[2] = NS;
89     frame[3] = frame[1]^frame[2];
90
91     new_bufSize = stuffing(frame, new_bufSize);
92     if(new_bufSize == -1) return -1;
93
94     /*imprime(frame, new_bufSize);
95     /*PARTE 2 - ENVIAR O FRAME***** */
96     //printf("Sending... ");
97     //sleep(1);
98     int res = write(fd, frame, new_bufSize);
99     if(res == -1) return -1;
100    stats_.num_frames++;
101    stats_.num_bytes += res;
102
103    /*PARTE 3 - LER A RESPOSTA DO RECEIVER E AGIR EM FUNCAO DISSO***** */
104    /* Se em (timeOut_Transmited) segundos nao receber nada reenviar
105    // Se receber REJ reenviar
106    // O reenvio so deve ser feito um total de numTries_Transmited vezes, aps isso return ERRO (-1)
107    // Se receber ACK correto retornar SUCESSO (numero de bytes enviados (>0))
108    //printf("C: %02x\n", frame[2]);
109    /* Listen to Receiver's answer */
110    //printf("Now im waiting ur answer \n");
111    (void) signal(SIGALRM, timeout_);
112    int ciclo = 1;
113    alarm(0);
114    while (STOP==FALSE) /* loop for input */
115    {
116        if(timeouts < numTries_Transmited){
117            if(flag_){
118                alarm(timeOut_Transmited);
119                flag_ = 0;
120                if(timeouts!=0){
121                    // after 1st timeout
122                    //printf("Sending again due to timeout. \n");
123                    res = write(fd, frame, new_bufSize); // send all the frame again
124                    if(res == -1) return -1;
125                    //printf("reenvion\n");
126                    stats_.num_frames++;
127                    stats_.num_bytes += new_bufSize;
128                    stats_.num_databytes += bufSize;
129                }
130            }
131        }

```

```

132     else{
133         alarm(0);
134         error = 1;
135         STOP = TRUE;
136     }
137
138     res = read(fd, &AUX_1, 1); //printf("read: %02x\n", AUX_1);
139     if(res == -1) return -1;
140     if(res == 0){
141         //printf("Timeout\n");
142     }
143
144     answer[STATE] = AUX_1;
145     switch (STATE)
146     {
147     case STATE0:
148         if(AUX_1 == FLAG) STATE = STATE1;
149         else STATE = STATE0;
150         break;
151     case STATE1:
152         if(AUX_1 == FLAG) STATE = STATE1;
153         else if(AUX_1 == A_1) STATE = STATE2;
154         else STATE = STATE0;
155         break;
156     case STATE2:
157         if(AUX_1 == FLAG) STATE = STATE1;
158         else if((AUX_1 == C_RR0)|| (AUX_1 == C_RR1)){ // "vai ler um RR"
159             if((Ns == C_S0)&&(AUX_1 == C_RR1)) STATE = STATE3;
160             else if((Ns == C_S1)&&(AUX_1 == C_RR0)) STATE = STATE3;
161             else STATE = STATE0;
162         }
163         else if((AUX_1 == C_REJ0)|| (AUX_1 == C_REJ1)){ // "vai ler um REJ"
164             if((Ns == C_S0)&&(AUX_1 == C_REJ0)) STATE = STATE6;
165             else if((Ns == C_S1)&&(AUX_1 == C_REJ1)) STATE = STATE6;
166             else STATE = STATE0;
167         }
168         else STATE = STATE0;
169         break;
170     case STATE3:
171         if(AUX_1 == FLAG) STATE = STATE1;
172         else if((AUX_1 == (C_RR0^A_1))|| (AUX_1 == (C_RR1^A_1))) STATE = STATE4;
173         else STATE = STATE0;
174         break;
175     case STATE4:
176         if(AUX_1 == FLAG) STATE = STATES;
177         else STATE = STATE0;
178         break;
179     case STATE5:
180         STOP = TRUE;
181         Ns = toggleNs(Ns);
182         break;
183     case STATE6:
184         if(AUX_1 == FLAG) STATE = STATE1;
185         else if((AUX_1 == (C_REJ0^A_1))|| (AUX_1 == (C_REJ1^A_1))) STATE = STATE7;
186         else STATE = STATE0;
187         break;
188     case STATE7:
189         if(AUX_1 == FLAG) STATE = STATE8;
190         else STATE = STATE0;
191         break;
192     case STATE8:
193         //Recebeu um REJ. Reenvia e volta ao estado
194         alarm(0);
195         res = write(fd, frame, new_bufSize); // send all the frame again
196         if(res == -1) return -1;
197         printf("REJ Recebido! A reenviar pacote \n");

```

```

198     |         timeouts++;
199     |         stats_.num_frames++;
200     |         stats_.num_bytes += new_bufSize;
201     |         stats_.num_databytes += bufSize;
202     |         stats_.num_R3J++;
203     |         STATE = STATE0;
204     |         break;
205     |
206     |     //printf("STATE: %d . AUX: %02x\n", STATE, AUX_1);
207     |     if(STATE == STATES) {
208     |         printf("RR Recebido\n");
209     |         Ns = toggleNs(Ns);
210     |         break;
211     |
212     }
213
214     }
215     alarm(0);
216
217     if(error) return -1;
218
219     //printf("\nretornei: %d\n", new_bufSize);
220     return new_bufSize;
221 }
222
223
224
225
226 // Receive data in packet
227 int lread(char* packet){
228 // controlo dos parametros
229 if(!packet) return -1;
230
231 /* Reinicar timeouts */
232 timeouts = 0;
233
234 conta = 1; // PARA AS ESTATISTICAS
235 /* Leitura */
236 int res, STOP = FALSE, STATE = 0, count=0, packetSize = 0, flag_encontrada = 0, packetSize_aux = 0;
237 char AUX[255], AUX_1;
238 char REJ0[] = {FLAG, A_1, C_REJ0, (A_1^C_REJ0), FLAG};
239 char REJ1[] = {FLAG, A_1, C_REJ1, (A_1^C_REJ1), FLAG};
240 char RR0[] = {FLAG, A_1, C_RR0, (A_1^C_RR0), FLAG};
241 char RR1[] = {FLAG, A_1, C_RR1, (A_1^C_RR1), FLAG};
242 char BCC_Check, C_Check;
243
244 /**
245 (void) signal(SIGALRM, timeout_);
246
247 while (STOP==FALSE) /* loop for input */
248 {
249     if(STATE!=STATES)
250         res = read(fd, &AUX_1, 1);
251     //printf("%02x ", AUX_1);
252     if(res == -1)
253         if(res == 0)
254             printf("timeout\n");
255     packet[packetSize] = AUX_1;
256     switch (STATE)
257     {
258     case STATE0:
259         if(AUX_1 == FLAG){
260             packetSize++;
261             STATE = STATES;
262

```

```

263     }
264     else packetSize++;
265
266     break;
267
268     case STATE0:
269         if(AUX_1 == FLAG){
270             packetSize = 1;
271             STATE = STATE1;
272         }
273         else{
274             packetSize = 0;
275             STATE = STATE0;
276         }
277         break;
278     case STATE1:
279         if(AUX_1 == FLAG){
280             packetSize = 1;
281             STATE = STATE1;
282         }
283         else if(AUX_1 == A_1){
284             packetSize = 2;
285             STATE = STATE2;
286         }
287         else{
288             packetSize = 0;
289             STATE = STATE0;
290         }
291         break;
292
293     case STATE2:
294         if(AUX_1 == FLAG){
295             packetSize = 1;
296             STATE = STATE1;
297         }
298         else if((AUX_1 == C_S0) || (AUX_1 == C_S1)){
299             packetSize = 3;
300             STATE = STATE3;
301             if(AUX_1 == toggleNs(Ns)){
302                 printf("PACOTE REPETIDO DETETADO. A REENVIAR RR\n");
303                 if(AUX_1 == C_S0) res = write(fd, RR1, 5);
304                 else if(AUX_1 == C_S1) res = write(fd, RR0, 5);
305                 packetSize = 0;
306                 STATE = STATE0;
307             }
308             else{
309                 packetSize = 0;
310                 STATE = STATE0;
311             }
312             break;
313
314     case STATE3:
315         C_Check = packet[2];
316         if(AUX_1 == FLAG){
317             packetSize = 1;
318             STATE = STATE1;
319         }
320         else if(AUX_1 == (A_1^Ns)){
321             // LEITURA DO CABEÇALHO COMPLETA PODE PASSAR PARA A LEITURA DOS DADOS
322             packetSize = 4;
323             STATE = STATE4;
324         }
325         else{
326             packetSize = 0;
327             STATE = STATE0;
328         }
329         break;
330     case STATE4:
331         //printf("\n");
332         C_Check = packet[2];
333         BCC_Check = packet[packetSize-2];
334         packetSize_aux = packetSize;
335
336         packetSize = resetFrameFormat(packet, packetSize);
337         if(packetSize == -1) return -1;
338         packetSize = destuffing(packet, packetSize);
339         if(packetSize == -1) return -1;
340
341         /*if(x1){
342             /printf("hehehehe\n");
343             x1 = 1;
344             packet[10] = 0x22;
345         }*/
346
347         // Atualiza as estatísticas
348         stats_.num_frames++;
349         stats_.num_databytes+=packetSize;
350         stats_.num_bytes+=packetSize_aux;
351
352         if(calculaBCC(packet, packetSize) == BCC_Check){ //caso em que recebe tudo direitinho calculaBCC(packet, packetSize) == BCC_Check (debugging propose)
353             printf("Everything seems cool.\n");
354             //envia RR a confirmar que está pronto a receber o proximo pacote
355             if(C_Check == C_S0) res = write(fd, RR1, 5);
356             else if(C_Check == C_S1) res = write(fd, RR0, 5);
357             if(res == -1) return -1;
358
359             Ns = toggleNs(Ns);
360         }
361         else{
362             //caso em que identifica que o pacote veio malificado
363             printf("Something went wrong here [ Sending REJ...\n");
364             if(C_Check == C_S0) res = write(fd, REJ0, 5);
365             else if(C_Check == C_S1) res = write(fd, REJ1, 5);
366             if(res == -1) return -1;
367             STATE = STATE0;
368             packetSize = 0;
369             stats_.num_REJ++;
370         }
371         STOP = TRUE;
372         break;
373     }
374
375 }
376
377 //imprime(packet, packetSize);
378 return packetSize;
379 }
// Closes previously opened connection; If showStatistics==TRUE, link layer should print statistics in the console on close
380 int llclose(int showStatistics){
381
382     /* Recalhar timeouts */
383     timeouts = 0, fFlag_ = 1;
384
385     (void) signal(SIGALRM, timeout_);
386
387     conta = 0; // PARA AS ESTATÍSTICAS
388     if(TYPE == RECEIVER){
389         //Espera o DISC
390         int res, STOP = FALSE, STATE = 0, error = 0;
391         char AUX[3], AUX_1;
392

```

```

394     char UA[] = {FLAG, A_1, C_UA, BCC_2, FLAG}; // Definição do UA segundo o protocolo
395     char DISC_rx[] = {FLAG, A_2, C_DISC, (A_2^C_DISC), FLAG};
396
397     while(STOP == FALSE){
398
399         /*if(timeout < numTries_Receiver){
400             if(flag_){
401                 alarm(timeOut_Receiver);
402                 flag_ = 0;
403             }
404         }
405         else{
406             alarm(0);
407             STOP = TRUE;
408             error = 1;
409             break;
410         }*/
411         res = read(fd, &AUX_1, 1); //printf("0x%02X STATE: %d\n", AUX_1, STATE);
412         AUX[STATE] = AUX_1;
413         switch (STATE)
414         {
415             case (STATE0):
416                 if(AUX_1 == FLAG) STATE = STATE1;
417                 else STATE = STATE0;
418                 break;
419             case(STATE1):
420                 if(AUX_1 == A_1) STATE = STATE2;
421                 else if(AUX_1 == FLAG) STATE = STATE1;
422                 else STATE = STATE0;
423                 break;
424             case(STATE2):
425                 if(AUX_1 == C_DISC) STATE = STATE3;
426                 else if(AUX_1 == FLAG) STATE = STATE1;
427                 else STATE = STATE0;
428                 break;
429             case(STATE3):
430                 if(AUX_1 == (A_1^C_DISC)) STATE = STATE4;
431                 else if(AUX_1 == FLAG) STATE = STATE1;
432                 else STATE = STATE0;
433                 break;
434             case(STATE4):
435                 if(AUX_1 == FLAG) STATE = STATE5;
436                 else STATE = STATE0;
437                 break;
438         }
439
440         if(STATE == STATE5) STOP = TRUE;
441     }
442     /*alarm(0);
443
444     if(error){
445         printf("DISC NOT FOUND\n");
446         return -1;
447     }
448 */
449     printf("DISC (recebido): (0x%02X)(0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", AUX[0], AUX[1], AUX[2], AUX[3], AUX[4]);
450
451     //Envia DISC
452     //sleep(15); // <-- Testes
453     res = write(fd, DISC_rx, 5);
454
455     if(res == -1) return -1;
456
457     printf("DISC (enviado): (0x%02X)(0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", DISC_rx[0], DISC_rx[1], DISC_rx[2], DISC_rx[3], DISC_rx[4]);
458
459     /* RECEBE UA */

```

```

458     /* RECEBE UA */
459     STOP = FALSE, STATE = 0, timeouts = 0, flag_ = 1;
460     while(STOP == FALSE){
461
462         /*if(timeouts < numTries_Receiver){
463             if(flag_){
464                 alarm(timeOut_Receiver);
465                 flag_ = 0;
466                 if(timeout != 0){
467                     res = write(fd, DISC_rx, 5);
468                     if(res == -1) return -1;
469                 }
470             }
471         }else{
472             alarm(0);
473             STOP = TRUE;
474             error = 1;
475             break;
476         }*/
477         res = read(fd, &AUX_1, 1);
478         AUX[STATE] = AUX_1;
479         switch (STATE)
480         {
481             case (STATE0):
482                 if(AUX_1 == FLAG) STATE = STATE1;
483                 else STATE = STATE0;
484                 break;
485             case(STATE1):
486                 if(AUX_1 == A_2) STATE = STATE2;
487                 else if(AUX_1 == FLAG) STATE = STATE1;
488                 else STATE = STATE0;
489                 break;
490             case(STATE2):
491                 if(AUX_1 == C_UA) STATE = STATE3;
492                 else if(AUX_1 == FLAG) STATE = STATE1;
493                 else STATE = STATE0;
494                 break;
495             case(STATE3):
496                 if(AUX_1 == (A_2^C_UA)) STATE = STATE4;
497                 else if(AUX_1 == FLAG) STATE = STATE1;
498                 else STATE = STATE0;
499                 break;
500             case(STATE4):
501                 if(AUX_1 == FLAG) STATE = STATES;
502                 else STATE = STATE0;
503                 break;
504             case(STATE5):
505                 if(STATE == STATES) STOP = TRUE;
506             }
507         if(STOP == TRUE){
508             /*alarm(0);
509             if(error){
510                 printf("UA NOT FOUND\n");
511                 return -1;
512             }*/
513             printf("UA (recebido): (0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", AUX[0], AUX[1], AUX[2], AUX[3], AUX[4]);
514
515             //mata-se
516             res = closeSerialTerminal(fd);
517             if(res == -1) return -1;
518             if(showStatistics) Statistics(stats_);
519             return 1;
520         }
521     }
522     else if(TYPE == TRANSMITTER){
523
524         int res, STOP = FALSE, STATE = 0, error = 0;
525         char AUX[5], AUX_1;
526
527
528         char UA[] = {FLAG, A_2, C_UA, (C_UA^A_2), FLAG}; // Definição do UA segundo o protocolo
529         char DISC_tx[] = {FLAG, A_1, C_DISC, (A_1^C_DISC), FLAG};
530         //Envia DISC
531         //sleep(5); // --> Testes
532         res = write(fd, DISC_tx, 5);
533         if(res == -1) return -1;
534
535         printf("DISC (enviado): (0x%02X)(0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", DISC_tx[0], DISC_tx[1], DISC_tx[2], DISC_tx[3], DISC_tx[4]);
536
537         //ESPERA DISC
538         while(STOP == FALSE){
539             if(timeouts < numTries_Transmitter){
540                 if(flag_){
541                     alarm(timeOut_Transmitter);
542                     flag_ = 0;
543                 }
544             }else{
545                 alarm(0);
546                 STOP = TRUE;
547                 error = 1;
548                 break;
549             }
550
551             res = read(fd, &AUX_1, 1);
552             AUX[STATE] = AUX_1;
553             switch (STATE)
554             {
555                 case (STATE0):
556                     if(AUX_1 == FLAG) STATE = STATE1;
557                     else STATE = STATE0;
558                     break;
559                 case(STATE1):
560                     if(AUX_1 == A_2) STATE = STATE2;
561                     else if(AUX_1 == FLAG) STATE = STATE1;
562                     else STATE = STATE0;
563                     break;
564                 case(STATE2):
565                     if(AUX_1 == C_DISC) STATE = STATE3;
566                     else if(AUX_1 == FLAG) STATE = STATE1;
567                     else STATE = STATE0;
568                     break;
569                 case(STATE3):
570                     if(AUX_1 == (A_2^C_DISC)) STATE = STATE4;
571                     else if(AUX_1 == FLAG) STATE = STATE1;
572                     else STATE = STATE0;
573                     break;
574                 case(STATE4):
575                     if(AUX_1 == FLAG) STATE = STATES;
576                     else STATE = STATE0;
577                     break;
578             }
579             if(STATE == STATES) STOP = TRUE;
580         }
581         if(STOP == TRUE){
582             /*alarm(0);
583             if(error){
584                 printf("DISC NOT FOUND\n");
585                 return -1;
586             }*/

```

```

579     }
580     if(STATE == STATES) STOP = TRUE;
581   }
582   alarm(0);
583   if(error){
584     printf("DISC NOT FOUND\n");
585     return -1;
586   }
587   printf("DISC (recebido): (0x%02X)(0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", AUX[0], AUX[1], AUX[2], AUX[3], AUX[4]);
588
589 // ENVIA UA
590 //MATA-se
591 res = write(fd, UA, 5);
592 if(res == -1) return -1;
593
594     printf("UA (Enviado): (0x%02X)(0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", UA[0], UA[1], UA[2], UA[3], UA[4]);
595
596 res = closeSerialTerminal(fd);
597 if(res == -1) return -1;
598 if(showStatistics) Statistics(stats_);
599
600 return 1;
601 }
602
603 return -1;
604 }

```

LinkLayer.h

```

1 #ifndef LINKLAYER
2 #define LINKLAYER
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 <signal.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <string.h>
13
14 typedef struct linkLayer{
15   char serialPort[50];
16   int role; //defines the role of the program: 0==Transmitter, 1=Receiver
17   int baudRate;
18   int numTries;
19   int timeOut;
20 } linkLayer;
21
22 //ROLE
23 #define NOT_DEFINED -1
24 #define TRANSMITTER 0
25 #define RECEIVER 1
26
27
28 //SIZE of maximum acceptable payload; maximum number of bytes that application layer should send to link layer
29 #define MAX_PAYLOAD_SIZE 1000
30
31 //CONNECTION deafault values
32 #define BAUDRATE_DEFAULT B38400
33 #define MAX_RETRANSMISSIONS_DEFAULT 3
34 #define TIMEOUT_DEFAULT 4
35 #define _POSIX_SOURCE 1 /* POSIX compliant source */
36
37 //MISC
38 #define FALSE 0
39 #define TRUE 1
40
41
42 // Opens a connection using the "port" parameters defined in struct linkLayer, returns "-1" on error and "1" on sucess
43 int llopen(linkLayer connectionParameters);
44 // Sends data in buf with size bufSize
45 int llwrite(char* buf, int bufSize);
46 // Receive data in packet
47 int llread(char* packet);
48 // Closes previously opened connection; if showStatistics==TRUE, link layer should print statistics in the console on close
49 int llclose(int showStatistics);
50
51
52 #endif
53
54
55
56

```

Receiver.c

```

1  #include "receiver.h"
2
3  /*
4   *Function: Opens the channel to communicate as receiver
5   *Return: returns 1 on sucess, -1 on error
6   */
7  int llopen_receiver(linkLayer connectionParameters, int fd){
8      // NONCANONICAL
9
10     unsigned char UA[] = {FLAG, A_2, C, BCC_2, FLAG}; // Definição do UA segundo o protocolo
11     int res;
12     int STOP = FALSE;
13     int STATE = 0;
14
15     /** Variaveis auxiliares ao while **/
16     aux - para determinar o indice do vetor
17     para - para determinar o fim da mensagem (na segunda flag)
18     AUX - vetor onde se guarda a mensagem
19     AUX_1 - onde se vai guardar a leitura que é feita 1 de cada vez
20
21
22     unsigned char AUX[255], AUX_1;
23     int aux = 0, para = 0;
24
25     while (STOP==FALSE) /* loop for input */
26     {
27         res = read(fd,&AUX_1,1);
28         AUX[STATE] = AUX_1;
29         //printf("AUX: %02X STATE: %d \n", AUX[STATE], STATE);
30         switch(STATE)
31         {
32             case (STATE0):
33                 if(AUX_1 == FLAG) STATE = STATE1;
34                 else STATE = STATE0;
35                 break;
36             case (STATE1):
37                 if(AUX_1 == A_1) STATE = STATE2;
38                 else if(AUX_1 == FLAG) STATE = STATE1;
39                 else STATE = STATE0;
40                 break;
41             case (STATE2):
42                 if(AUX_1 == C) STATE = STATE3;
43                 else if(AUX_1 == FLAG) STATE = STATE1;
44                 else STATE = STATE0;
45                 break;
46             case (STATE3):
47                 if(AUX_1 == BCC_1) STATE = STATE4;
48                 else if(AUX_1 == FLAG) STATE = STATE1;
49                 else STATE = STATE0;
50                 break;
51             case (STATE4):
52                 if(AUX_1 == FLAG) STATE = STATE5;
53                 else STATE = STATE0;
54                 break;
55         }
56
57
58         if (STATE == STATE5) STOP = TRUE;
59     }
60     printf("SET (Recebido): (0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", AUX[0], AUX[1], AUX[2], AUX[3], AUX[4]);
61
62     /* Envio do UA de modo a confirmar a receção do SET */
63     //sleep(4); // <-- tests
64     res = write(fd, UA, 5);
65     printf("UA (enviado): (0x%02X)(0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", UA[0], UA[1], UA[2], UA[3], UA[4]);
66
67
68     return 1;
69 }
70
71

```

Receiver.h

```

1  #ifndef RECEIVER_H
2  #define RECEIVER_H
3
4  #include "geral.h"
5
6  /*
7   *Function: Opens the channel to communicate as receiver
8   *Return: returns 1 on sucess, -1 on error
9   */
10  int llopen_receiver(linkLayer connectionParameters, int fd);
11
12  /*
13   *Function: Closes the communication as receiver
14   *Return: returns 1 on sucess, -1 on error
15   */
16  int llclose_receiver(linkLayer connectionParameters); // NAO SEI QUAIS SERAO OS PARAMETROS NECESSARIOS
17
18  #endif

```

Transmitter.c

```

1  #include "transmisor.h"
2
3  int n_timeouts=0, flag=1;
4  void timeout()           // atende alarme
5  {
6      printf("timeout #d...\n", (n_timeouts+1));
7      flag=1;
8      n_timeouts++;
9
10 }
11 /*
12 *Function: Opens the channel to communicate as transmisor
13 *Return: returns 1 on sucess, -1 on error
14 */
15 int llopen_transmisor(linkLayer connectionParameters, int fd){
16     // WRITE NONCANONICAL
17     n_timeouts=0, flag=1;
18     int res;
19
20     unsigned char SET[] = {FLAG, A_1, C, BCC_1, FLAG}; // Definicao do SET segundo o protocolo
21
22     int STOP=FALSE;
23     /* Envio do set atravez da função write */
24     res = write(fd,SET,5);
25     printf("SET (enviado): (0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", SET[0], SET[1], SET[2], SET[3], SET[4]);
26
27     /**
28      ## Variaveis auxiliares ao while ##
29
30      aux - para determinar o indice do vetor
31      para - para determinar o fim da mensagem (na segunda flag)
32      AUX - vetor onde se guardara a mensagem
33      AUX_1 - onde se vai guardar a leitura que é feita 1 de cada vez
34
35     */
36
37     int aux = 0, para = 0;
38     unsigned char AUX[255], AUX_1;
39     int STATE = 0, contador = 0;
40
41     (void) signal(SIGALRM, timeout); // instala rotina que atende interrupcao
42
43     while (STOP==FALSE) /* loop for input */
44     {
45         if((n_timeouts<connectionParameters.numTries)&&(STATE!=STATE5)){
46             if(flag){
47                 alarm(connectionParameters.timeOut);
48                 flag = 0;
49
50                 if(n_timeouts!=0){          // after 1st timeout
51                     res = write(fd,SET,5); // send SET again
52                 }
53             }
54             else{
55                 STOP=TRUE;
56                 alarm(0);
57             }
58
59             res = read(fd,&AUX_1,1);
60
61             AUX[STATE] = AUX_1;
62
63             switch(STATE)
64             {
65                 case (STATE0):
66
67                     if(AUX_1 == FLAG) STATE = STATE1;
68                     else STATE = STATE0;
69                     break;
70
71                 case (STATE1):
72                     if(AUX_1 == A_2) STATE = STATE2;
73                     else if(AUX_1 == FLAG) STATE = STATE1;
74                     else STATE = STATE0;
75                     break;
76
77                 case (STATE2):
78                     if(AUX_1 == C) STATE = STATE3;
79                     else if(AUX_1 == FLAG) STATE = STATE1;
80                     else STATE = STATE0;
81                     break;
82
83                 case (STATE3):
84                     if(AUX_1 == BCC_2) STATE = STATE4;
85                     else if(AUX_1 == FLAG) STATE = STATE1;
86                     else STATE = STATE0;
87                     break;
88
89             }
90
91             if (STATE == STATE5)
92             {
93                 STOP = TRUE;
94                 para = 1;
95                 contador = 0;
96                 alarm(0);
97             }
98
99             if(n_timeouts>=connectionParameters.numTries){
100                 printf("Failed to start connection...\n");
101                 return -1;
102             }
103             if(STATE==STATE5){
104                 printf("UA (recebido): (0x%02X)(0x%02X)(0x%02X)(0x%02X)(0x%02X)\n", AUX[0], AUX[1], AUX[2], AUX[3], AUX[4]);
105                 return 1;
106             }
107
108             // ERRO
109             return -1;
110
111     }

```

Transmitter.h

```
1  #ifndef TRANSMITTER_H
2  #define TRANSMITTER_H
3
4  #include "geral.h"
5
6  /*
7   *Function: Opens the channel to communicate as transmitter
8   *Return: returns 1 on sucess, -1 on error
9   */
10 int llopen_transmifter(linkLayer connectionParameters, int fd);
11 /*
12 *Function: Closes the communication as transmitter
13 *Return: returns 1 on sucess, -1 on error
14 */
15 int llclose_transmifter(linkLayer connectionParameters); // NAO SEI QUAIS SERAO OS PARAMETROS NECESSARIOS
16 /*
17 *
18 *Retorno: none
19 */
20 void timeout();
21
22 #endif
```

Geral.c

```
1  #include "linklayer.h"
2
3  #include <string.h>
4  #include <stdio.h>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8
9 /*
10  * $1 /dev/ttySxx
11  * $2 tx | rx
12  * $3 filename
13  */
14
15 int main(int argc, char *argv[]) {
16
17  if (argc < 4) {
18
19      printf("usage: programe /dev/ttySxx tx|rx filename\n");
20      exit(1);
21  }
22
23
24  printf("%s %s %s\n", argv[1], argv[2], argv[3]);
25  fflush(stdout);
26
27  if (strcmp(argv[2], "tx") == 0)
28  {
29      // *****
30      // tx mode
31      printf("tx mode\n");
32
33      // open connection
34      struct linkLayer ll;
35      sprintf(ll.serialPort, "%s", argv[1]);
36      ll.role = 0;
37      ll.baudRate = 9600;
38      ll.numTries = 3;
39      ll.timeOut = 3;
40
41      if(llopen(ll)==-1) {
42          fprintf(stderr, "Could not initialize link layer connection\n");
43          exit(1);
44      }
45
46      printf("connection opened\n");
47      fflush(stdout);
48      fflush(stderr);
49
50      // open file to read
51      char *file_path = argv[3];
52      int file_desc = open(file_path, O_RDONLY);
53      if(file_desc < 0) {
54          fprintf(stderr, "Error opening file: %s\n", file_path);
55          exit(1);
56      }
57
58      // cycle through
59      const int buf_size = MAX_PAYLOAD_SIZE-1;
60      unsigned char buffer[buf_size+1];
61      int write_result = 0;
62      int bytes_read = 1;
63
64      while (bytes_read > 0)
65      {
66          bytes_read = read(file_desc, buffer+1, buf_size);
67          if(bytes_read < 0) {
```

```

67         }
68     }
69     else if (bytes_read > 0) {
70         // continue sending data
71     buffer[0] = 1;
72     write_result = llwrite(buffer, bytes_read+1);
73     if(write_result < 0) {
74         fprintf(stderr, "Error sending data to link layer\n");
75         break;
76     }
77     printf("read from file -> write to link layer, %d\n", bytes_read);
78 }
79 else if (bytes_read == 0) {
80     // stop receiver
81     buffer[0] = 0;
82     llwrite(buffer, 1);
83     printf("App layer: done reading and sending file\n");
84     break;
85 }
86 }
87 sleep(1);
88 }
89 // close connection
90 llclose(1);
91 close(file_desc);
92 return 0;
93 }
94 }
95 else
96 {
97     // *****
98     // rx mode
99     printf("rx mode\n");
100
101     struct linkLayer ll;
102     sprintf(ll.serialPort, "%s", argv[1]);
103     ll.role = 1;
104     ll.baudRate = 9600;
105     ll.numTries = 3;
106     ll.timeOut = 3;
107
108     if(llopen(&ll)==-1) {
109         fprintf(stderr, "Could not initialize link layer connection\n");
110         exit(1);
111     }
112
113     char *file_path = argv[3];
114     int file_desc = open(file_path, O_RDWR|O_CREAT, S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
115     if(file_desc < 0) {
116         fprintf(stderr, "Error opening file: %s\n", file_path);
117         exit(1);
118     }
119
120     int bytes_read = 0;
121     int write_result = 0;
122     const int buf_size = MAX_PAYLOAD_SIZE;
123     unsigned char buffer[buf_size];
124     int total_bytes = 0;
125
126     while (bytes_read >= 0)
127     {
128         bytes_read = llread(buffer);
129         if(bytes_read < 0) {
130             fprintf(stderr, "Error receiving from link layer\n");
131             break;
132         }
133     }
134     else if (bytes_read > 0) {
135         if (buffer[0] == 1) {
136             write_result = write(file_desc, buffer+1, bytes_read-1);
137             if(write_result < 0) {
138                 fprintf(stderr, "Error writing to file\n");
139                 break;
140             }
141             total_bytes = total_bytes + write_result;
142         }
143         printf("read from file -> write to link layer, %d %d %d\n", bytes_read, write_result, total_bytes);
144     }
145     else if (buffer[0] == 0) {
146         printf("App layer: done receiving file\n");
147         break;
148     }
149 }
150 llclose(1);
151 close(file_desc);
152 return 0;
153 }
154 }
155 }
156 }
157 }
158 }
159 }
```