

Protocolo de Ligação de Dados

Redes de Computadores - 1º Trabalho Laboratorial

Turma 5a1530- Grupo G

[up201904814@edu.fe.up.pt](mailto:up201904814@edu.fe.up.pt) José Castro Baptista [up[201905296](https://sigarra.up.pt/feup/pt/fest_geral.cursos_list?pv_num_unico=201905296)@edu.fe.up.pt](mailto:up@edu.fe.up.pt) Tomás Vasconcelos Estácio

Abril, 2022

**Conteúdos**

[1 Sum´ario 3](#_Toc99376589)

[2 Introdu¸c˜ao 3](#_Toc99376590)

[3 Arquitetura 3](#_Toc99376591)

[4 Casos de Uso Principais 4](#_Toc99376592)

[4.1 Identifica¸c˜ao 4](#_Toc99376593)

[4.2 Sequˆencia de Chamadas 4](#_Toc99376594)

[5 Protocolo de Liga¸c˜ao L´ogica 4](#_Toc99376595)

[5.1 Leitura da Porta S´erie 5](#_Toc99376596)

[5.2 Escrita na Porta S´erie 7](#_Toc99376597)

[5.3 Interface para uso na camada de aplica¸c˜ao 7](#_Toc99376598)

[5.3.1 Estabelecimento da liga¸c˜ao 7](#_Toc99376599)

[5.3.2 T´ermino da conex˜ao 7](#_Toc99376600)

[5.3.3 Envio de dados 8](#_Toc99376601)

[5.3.4 Rece¸c˜ao de dados 8](#_Toc99376602)

[6 Protocolo de Aplica¸c˜ao 8](#_Toc99376603)

[6.1 Pacotes de Controlo 8](#_Toc99376604)

[6.2 Leitura e Escrita de dados 9](#_Toc99376605)

[6.3 Leitura do ficheiro e constru¸c˜ao dos pacotes 9](#_Toc99376606)

[6.4 Interpreta¸c˜ao dos pacotes e escrita do ficheiro 10](#_Toc99376607)

[7 Valida¸c˜ao 10](#_Toc99376608)

[8 Eficiˆencia 10](#_Toc99376609)

[8.1 Protocolo utilizado 10](#_Toc99376610)

[8.1.1 Funcionamento do protocolo 11](#_Toc99376611)

[8.1.2 Mecanismos de dete¸c˜ao de erros 11](#_Toc99376612)

[8.2 An´alise de Dados 11](#_Toc99376613)

[8.2.2 Gr´aficos 12](#_Toc99376614)

[9 Conclusão 14](#_Toc99376615)

[10 Anexo 1 15](#_Toc99376616)

# Sumário

No âmbito da unidade curricular de *Redes de Computadores*, do curso *Licenciatura em Engenharia Eletrotécnica e de Computadores*, foi realizado um trabalho com o objetivo de estudar a transferência de dados através de uma aplicação com recurso a um protocolo de ligação de dados.

Entre os conteúdos abordados destacam-se a Camada de Ligação de Dados (*Data Link Layer*) e a Camada Física (*Physical Layer*). A proposta de trabalho baseou-se em realizar uma transferência de ficheiros entre dois computadores (de forma assíncrona), através de uma ligação por cabo pelas portas de série RS-232 de cada máquina.

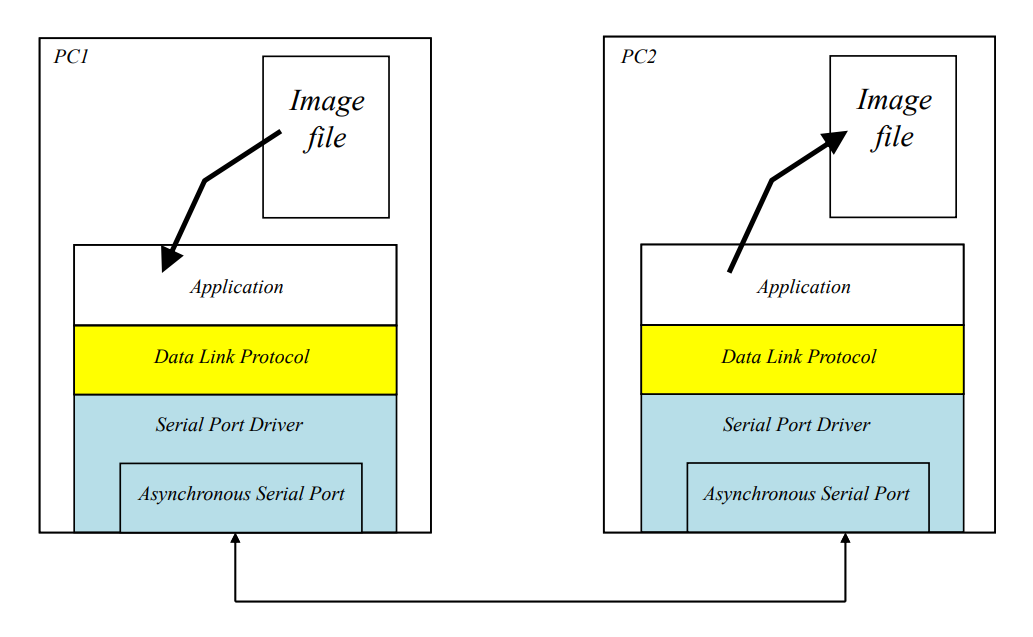
# Introdução

O relatório tem a seguinte estrutura:

1. **Arquitetura**: Funcionamento da aplicação e da interface do utilizador;
2. **Estrutura do código:** Demonstração das API’s, estruturas de dados utilizadas, principais funções e a sua conexão com a arquitetura;
3. **Casos de uso principais:** Identificação de casos de usos principais e sequência de chamadas de funções;
4. **Protocolo de ligação lógica:** Identificação dos principais aspetos funcionais e descrição da estratégia de implementação;
5. **Protocolo de aplicação**
6. **Validação:** Descrição dos testes efetuados com apresentação quantificada dos resultados;
7. **Eficiência do protocolo de ligação de dados:** Inferência estatística da eficiência do protocolo;
8. **Conclusão:** Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

# Arquitetura

A aplicação está dividida em três camadas (aplicação, ligação de dados e protocolo) e em dois modos de funcionamento, o emissor e o recetor. Estes dois modos são responsáveis por escrever e ler na porta série seguindo o protocolo de ligação de dados. Para além disso, o emissor é responsável pela leitura do ficheiro e processamento das tramas para envio. Já o recetor verifica as tramas recebidas e cria um ficheiro com a informação previamente recebida e processada. As camadas traduzem-se em módulos de código, mas os modos de funcionamento não. Isto acontece porque o utilizador utiliza o mesmo executável tanto para o emissor quanto para o recetor e o utilizador escolhe o modo de funcionamento utilizando argumentos de linha de comandos.

* A camada da aplicação é a responsável pela leitura/criação do novo ficheiro e pelo envio e receção da informação dos ficheiros, sendo estas funções provenientes de um nível superior à camada de ligação de dados. Esta chama as funções da camada da ligação de dados, para envio/receção da informação de dados, mantendo-se completamente independente da camada de nível inferior uma vez que desconhece os seus métodos de atuar.
* A camada da ligação de dados é a responsável tanto por ler e escrever na porta série, lidando com possíveis erros na transmissão das tramas. Esta camada não tem qualquer autonomia, só realiza as ações de transmissão e emissão quando solicitada.

É responsável pelo estabelecimento da ligação, torna o protocolo sólido, garantindo a sua consistência. Por este motivo, é considerada a camada de mais baixo nível do programa, sendo que trata da abertura da porta série, da transmissão de informação (escrita e leitura) e do seu posterior fecho. É também da sua responsabilidade testar se a informação foi escrita/recebida corretamente, através de byte stuffing e de testes de erros e segurança da trama, como os cálculos do BCC.

* A camada de protocolo já está implementada e a interação é feita através de system calls.

# Estrutura do código

O código foi estruturado de forma a tentar otimizar tanto as funções como as variáveis partilhadas dentro de cada processo. Devido às diversas funções internas que se foram desenvolvendo ao longo do projeto, criou-se um ficheiro *constants.h* que inclui todas estas, a serem utilizadas pelas funções do protocolo de ligação de dados.

* main, fornecida pelo professor, abre as ligações em cada terminal:

Com o argumento “TX”, inicia a aplicação em modo de transmissão;

É criada uma struct ‘linkLayer’ que guarda as informações de função, baudrate, número de tentativas de reconexão, e a duração de um alarme; (o struct tem baudrate, numtries, timeout, role tx ou rx, e definições da serial port)

Abre o ficheiro a ser enviado;

Corre um ciclo contínuo onde são enviados payloads de um certo tamanho especificado para a camada link;

O buffer que armazena payloads também terá um byte a sinalizar o estado da transmissão do ficheiro na posição 0 (se estiver a 0, termina o envio do ficheiro, se estiver a 1, continua o envio do ficheiro).

* linklayer.h – é um ficheiro header, fornecido pelos docentes, com as especificações e argumentos das funções a implementar na camada de link. É aqui onde estão as declarações de llopen, llclose, llread e llwrite, a struct ‘linkLayer’, e outras constantes úteis.

Estrutura da Link Layer:

typedef struct linkLayer{

    char serialPort[50];

    int role; //defines the role of the program: 0==Transmitter, 1=Receiver

    int baudRate;

    int numTries;

    int timeOut;

} linkLayer;

**Principais funções:**

void control\_alarm()

speed\_t get\_baud(int baud)

**int llopen(linkLayer connectionParameters)**

int establishment\_trans()

unsigned char informationcheck()

int transmitter\_information\_write(char\* buf, int bufSize)

int transmitter\_information\_read()

int termination\_trans()

int establishment\_rec()

unsigned char confirmationcheck()

int receiver\_information\_read(char\* packet)

int receiver\_information\_write(char\* packet)

int termination\_rec()

**int llwrite(char\* buf, int bufSize)**

**int llread(char\* packet)**

**int llclose(int showStatistics)**

**Macros**

* BAUDRATE: usado na struct termios, durante o llopen(), indica a capacidade da ligação.
* RECEIVER a 0 e TRANSMITTER A 1: para efeitos de distinção do papel da aplicação.
* TIMEOUT: tempo, em segundos, que o emissor deve esperar resposta do recetor antes de reenviar a trama.
* MAX\_SIZE: indica o tamanho máximo, em bytes, de informação do ficheiro que cabe num pacote da aplicação.

# Casos de Uso Principais

# 

Os casos de uso principais são:

* a interface através da linha de comandos, onde o utilizador seleciona que função o seu dispositivo deve ter, que porta utilizará para a comunicação e que ficheiro quer enviar;
* a transmissão do ficheiro através da porta série, conseguindo ainda visualizar estatísticas do desempenho da transferência do mesmo. O utilizador deverá então introduzir na linha de comandos o nome do programa, e de seguida, como argumentos, a porta a ser utilizada, o papel do processo, rx ou tx, e o ficheiro que pretende enviar (caso seja o transmissor). A sequência das funções executadas é:

1. Emissor escolhe o ficheiro a enviar, na interface. (tx)

2. Configuração da ligação entre os dois computadores.

3. Estabelecimento da ligação.

4. Transmissor envia os dados utilizando o mecanismo Stop-and-Wait . (rx)

5. Recetor recebe os dados e envia tramas de supervisão RR e REJ

6. Recetor guarda os dados num ficheiro com o mesmo nome do ficheiro

enviado pelo emissor.

7. Terminação da ligação..

Sequência de chamada das principais funções:

**llopen**: abertura da porta de série para escrita e leitura.

**llread**: receção de trama de informação: se se verificar algum erro no cabeçalho, no campo de dados ou no número de sequência é enviada uma trama de supervisão REJ e a execução permanece num ciclo até a receção não detetar nenhum erro, sendo enviada uma trama de supervisão RR nesse caso.

**llwrite**: envio de trama de informação com o pacote inserido no campo de dados; receção de trama de supervisão RR/REJ (com recurso a função receiveRRREJ (????)). Por cada REJ recebido é retransmitida a trama de informação. O mesmo acontece se o transmissor não tiver recebido nenhuma trama de supervisão em relação à trama enviada durante três segundos.

**llclose**: envio de trama não numerada DISC, receção de trama não numerada DISC, envio de trama não numerada UA e fecho da porta de série.

# Protocolo de Ligação Lógica

**Aspetos fundamentais e estratégia de implementação**

* **Estabelecimento e terminação da ligação**

A função ***establishment\_trans*** é responsável por estabelecer a ligação enviando uma trama SET e recebendo uma trama UA (modo transmissor) ou a função ***establishment\_rec*** recebendo uma trama SET e enviando uma trama UA (modo recetor).

* **Enviar e receber informação através da porta de série recorrendo a *framing***

As funções ***llwrite*** e ***llread*** fornecem à camada da aplicação esses serviços.

* **Numeração de tramas**

O transmissor/recetor nesta camada sabe a todo o momento o número de sequência da trama que vai enviar/receber, e vai atualizando-o convenientemente: ver o anexo IV.

* **Controlo de erros**

Ao receber uma trama é feita a verificação do BCC do cabeçalho, do BCC do campo de dados (se aplicável e após *destuffing*) e do número de sequência. Exemplo destas verificações do lado do recetor.

* **Confirmação**

Após processar uma trama, o recetor envia uma confirmação positiva/negativa (através de tramas RR ou REJ) consoante a verificação da trama e o número de sequência: ver o anexo VI.

* ***Stuffing* e *destuffing* dos pacotes da camada da aplicação**

O byte stuffing e destuffing baseia-se em simples verificações, realizada byte a byte sobre os bytes de informação recebidos (e BCC2 também), que os compara aos bytes especiais, nomeadamente a FLAG (0x7e) e o ESC (0x7d). No caso do byte stuffing, estes bytes são transformados numa sequência de 2 bytes: 0x7d 0x5e para o primeiro e 0x7d e 0x5d para o segundo. No byte destuffing, é efetuado o processo inverso. Desta forma, no caso de algum dos bytes de informação ou o BCC2 coincidir com estes bytes especiais, nunca serão confundidos com os bytes delimitadores da trama, pelo que a informação é transmitida corretamente.

* **Retransmissões**

Na nossa aplicação foi usado um protocolo baseado no protocolo *Stop-and-Wait* para controlo de erros. Quando o emissor manda qualquer tipo de tramas (U, S ou I) espera uma resposta. Essa resposta é **RR** caso o recetor receba os dados sem erros, e **REJ** caso contrário. Assim, o emissor sabe se deve mandar uma nova trama ou reenviar a mesma. O *Nr* destas tramas de resposta varia conforme o emissor tenha enviado uma trama de *Ns* 0 ou 1, para este, no futuro, saber que trama deve mandar e para ajudar no tratamento de duplicados.

|  |  |  |
| --- | --- | --- |
|  | Resposta do Recetor | |
| Trama enviada pelo Emissor | **Sem erros** | **Com erros** |
| Ns=0 | RR (Nr=1) | REJ (Nr=0) |
| Ns=1 | RR (Nr=0) | REJ (Nr=1) |

# Protocolo de Aplicação

# A camada de aplicação, ao contrário da camada de ligação lógica, não comunica com a porta série e está encarregue de ler do ficheiro a enviar e escrever o ficheiro recebido, criação e processamento de pacotes de dados e divisão da informação do ficheiro conforme o valor de MAX\_PAYLOAD\_SIZE, e construção dos pacotes de dados, respeitando a sua estrutura.

# Leitura do ficheiro a enviar, por parte do emissor, e criação/escrita de um ficheiro destino por parte do recetor.

# 

# Validação e eficiência

De forma a estudar a aplicação desenvolvida, foram efetuados os seguintes testes:

* Envio de ficheiros de vários tamanhos.
* Introdução de ruído enquanto se envia um ficheiro.
* Interrupção da ligação por alguns segundos enquanto se envia um ficheiro.
* Envio do um ficheiro com variação do tamanho de pacotes.
* Envio do um ficheiro com variação das capacidades de ligação (*baudrate*).

Todos os testes foram concluídos com sucesso.

Definições:

• S – eficiência – R/(tprop+R+tprop)

• R - débito recebido

• Tprop - tempo de propagação da trama

**Tempo de propagação**: O próximo gráfico mostra a variação da eficiência consoante o tempo de propagação, o que valida as fórmulas no sentido de que S varia em proporcionalidade inversa com a.

**Baudrate**: O gráfico permite-nos concluir que a eficiência do protocolo diminui com o aumento da capacidade da ligação de dados. Estima-se que esta viesse a subir até um dado valor. No entanto, infelizmente, essa subida não está representada na gama de valores que foram recolhidos.

Valores default:

Baudrate=38400

Framesize=1000B

Tprop= 0

# Conclusão

# Anexo 1 – Código fonte

# Modules

# application

constants.h –

linklayer.c-

linklayer.h-

main.c-

linklayer.c

#include "constants.h"

#include "linklayer.h"

#include <signal.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/un.h>

#include <fcntl.h>

#include <termios.h>

#include <stdio.h>

#include <unistd.h>

struct termios oldtio, newtio;

int tx=0;

int rx=0;

int tentat=0;

int tentatREJ=0;

int RETRANS=0;

int status=0;

static int fd;

int duplicado=0;

int Ns\_anterior\_trans=1;

int Ns\_trans=0;

int Nr\_anterior\_rec=0;

int Nr\_rec=1;

int Ns\_anterior\_rec=1;

int Ns\_rec=0;

int Nr\_anterior\_trans=0;

int Nr\_trans=1;

//statistics counters

int TOTALREAD\_TRANS, TOTALWRITE\_TRANS;

int TOTALREAD\_REC, TOTALWRITE\_REC;

int rej\_count\_trans=0;

int rej\_count\_rec=0;

int rr\_count\_trans=0;

int rr\_count\_rec=0;

int error\_count=0;

int resent\_write=0;

int dup\_count\_trans=0;

int NUMTRIES;

int TIMEOUT;

unsigned char BCC2\_inicial = 0x00;

unsigned char BCC2\_final = 0x00;

volatile int STOP=FALSE;

void control\_alarm(){

  printf("TIMEOUT #%d\n", tentat+1);

  tentat++;

  STOP = FALSE;

  return;

}

speed\_t get\_baud(int baud)

{

    switch (baud) {

    case 9600:

        return B9600;

    case 19200:

        return B19200;

    case 38400:

        return B38400;

    case 57600:

        return B57600;

    case 115200:

        return B115200;

    case 230400:

        return B230400;

    case 460800:

        return B460800;

    case 500000:

        return B500000;

    case 576000:

        return B576000;

    case 921600:

        return B921600;

    case 1000000:

        return B1000000;

    case 1152000:

        return B1152000;

    case 1500000:

        return B1500000;

    case 2000000:

        return B2000000;

    case 2500000:

        return B2500000;

    case 3000000:

        return B3000000;

    case 3500000:

        return B3500000;

    case 4000000:

        return B4000000;

    default:

        return B0;

    }

}

int llopen(linkLayer connectionParameters)

{

    NUMTRIES = connectionParameters.numTries;

    TIMEOUT = connectionParameters.timeOut;

    if(connectionParameters.role == 0) //transmitter

    {

        tx = 1;

        fd = open(connectionParameters.serialPort, O\_RDWR | O\_NOCTTY );

        if(fd <0){

            perror(connectionParameters.serialPort);

            exit(-1);

        }

        if(tcgetattr(fd,&oldtio) == -1){ /\* save current port settings \*/

            perror("tcgetattr");

            exit(-1);

        }

        bzero(&newtio, sizeof(newtio));

        newtio.c\_cflag = get\_baud(connectionParameters.baudRate) | CS8 | CLOCAL | CREAD;

        newtio.c\_iflag = IGNPAR;

        newtio.c\_oflag = 0;

        /\* set input mode (non-canonical, no echo,...) \*/

        newtio.c\_lflag = 0;

        newtio.c\_cc[VTIME]    = connectionParameters.timeOut\*10;   /\* inter-character timer unused \*/

        newtio.c\_cc[VMIN]     = 0;   /\* blocking read until 5 chars received \*/

        tcflush(fd, TCIOFLUSH);

        if(tcsetattr(fd,TCSANOW,&newtio) == -1){

            perror("tcsetattr");

            exit(-1);

        }

        int check = establishment\_trans();

        if(check == 1) return 1;

        else{

            printf("Error trying to open the connection (tx)\n");

            llclose(FALSE);

            return -1;

        }

    }

    else if(connectionParameters.role == 1) //receiver

    {

        rx = 1;

        fd = open(connectionParameters.serialPort, O\_RDWR | O\_NOCTTY );

        if(fd <0){

            perror(connectionParameters.serialPort);

            exit(-1);

        }

        if(tcgetattr(fd,&oldtio) == -1){ /\* save current port settings \*/

            perror("tcgetattr");

            exit(-1);

        }

        bzero(&newtio, sizeof(newtio));

        newtio.c\_cflag = get\_baud(connectionParameters.baudRate) | CS8 | CLOCAL | CREAD;

        newtio.c\_iflag = IGNPAR;

        newtio.c\_oflag = 0;

        /\* set input mode (non-canonical, no echo,...) \*/

        newtio.c\_lflag = 0;

        newtio.c\_cc[VTIME]    = connectionParameters.timeOut\*10;   /\* inter-character timer unused \*/

        newtio.c\_cc[VMIN]     = 0;   /\* blocking read until 1 chars received \*/

        tcflush(fd, TCIOFLUSH);

        if(tcsetattr(fd,TCSANOW,&newtio) == -1){

            perror("tcsetattr");

            exit(-1);

        }

        int check = establishment\_rec();

        if(check == 1) return 1;

        else{

            printf("Error trying to open the connection (rx)\n");

            llclose(FALSE);

            return -1;

        }

    }

    else return -1;

}

int establishment\_trans()

{

    int state = 0;

    int total = 0;

    int res = 0;

    unsigned char inicio[5];

    unsigned char aux;

    inicio[0] = FLAG;

    inicio[1] = A\_TRANS;

    inicio[2] = SET;

    inicio[3] = (A\_TRANS^SET);

    inicio[4] = FLAG;

    (void) signal(SIGALRM, control\_alarm);

    while(tentat < NUMTRIES && state != 5)

    {

        if(STOP == FALSE){

            alarm(TIMEOUT);

            STOP = TRUE;

            res = write(fd, inicio, 5);

            printf("SET: 0x%X | 0x%X | 0x%X | 0x%X | 0x%X\n", inicio[0], inicio[1], inicio[2], inicio[3], inicio[4]);

        }

        res = read(fd, &aux, 1);

        total += res;

        switch(state){

            case 0:

            if(aux == FLAG) state = 1;

            else total = 0;

            break;

            case 1:

            if(aux == A\_REC) state = 2;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0,

                total = 0;

            }

            break;

            case 2:

            if(aux == UA) state = 3;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 3:

            if(aux == (A\_REC^UA)) state = 4;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 4:

            if(aux == FLAG) state = 5;

            else{

                state = 0;

                total = 0;

            }

            break;

        }

    }

    if(state == 5){

        (void) signal(SIGALRM, SIG\_IGN);

        STOP = FALSE;

        tentat = 0;

        return 1;

    }

    else return -1;

}

unsigned char informationcheck()

{

    if(Ns\_trans == 0) return 0x00;

    else return 0x02;

}

int transmitter\_information\_write(char\* buf, int bufSize)

{

    int res = 0;

    int state = 0;

    int total = 0;

    unsigned char trama[2\*bufSize + 7];

    unsigned char aux;

    unsigned char BCC2 = 0x00;

    int j = 3;

    //stuffing (FLAG -> ESC e FLAG^0x20 ; ESC -> ESC e ESC^0x20)

    trama[0] = FLAG;

    trama[1] = A\_TRANS;

    trama[2] = informationcheck(buf, bufSize);

    for(int i=0; i<bufSize; i++)

    {

        BCC2^=buf[i];

    }

    for(int i=0; i<bufSize; i++)

    {

        //BCC2 ^= buf[i];

        if(buf[i] == FLAG){

            trama[j] = ESC;

            j++;

            trama[j] = 0x5E;

        }

        else if(buf[i] == ESC){

            trama[j] = ESC;

            j++;

            trama[j] = 0x5D;

        }

        else trama[j] = buf[i];

        j++;

    }

    if(BCC2 == FLAG){

        trama[j] = ESC;

        j++;

        trama[j] = (FLAG^0x20);

    }

    else if(BCC2 == ESC){

        trama[j] = ESC;

        j++;

        trama[j] = (ESC^0x20);

    }

    else trama[j] = BCC2;

    j++;

    trama[j] = FLAG;

    total = write(fd, trama, j+1);

    if(total > 0) return bufSize;

    else return 0;

}

int transmitter\_information\_read()

{

    unsigned char aux;

    int state = 0;

    int res = 0;

    int total = 0;

    while(state != 6)

    {

        res = read(fd, &aux, 1);

        if(res == 0) return 0;

        total += res;

        switch(state){

            case 0:

            if(aux == FLAG) state = 1;

            else total = 0;

            break;

            case 1:

            if(aux == A\_REC) state = 2;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 2:

            if(aux == 0x01 || aux == 0x21){

                if(aux == 0x01 && Nr\_anterior\_trans == 0){

                    duplicado = 1;

                    state = 3;

                }

                else if(aux == 0x21 && Nr\_anterior\_trans == 1){

                    duplicado = 1;

                    state = 3;

                }

                else state = 3;

            }

            else if(aux == 0x05 || aux == 0x25){

                state = 4;

                RETRANS = 1;

            }

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 3:

            if(aux == (A\_REC^0x01) || aux == (A\_REC^0x21)) state = 5;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 4:

            if(aux == (A\_REC^0x05) || aux == (A\_REC^0x25)) state = 5;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 5:

            if(aux == FLAG) state = 6;

            else{

                state = 0;

                total = 0;

            }

            break;

        }

        if(RETRANS == 1 && state == 6){

            RETRANS = 0;

            return -1;

        }

    }

    return 1;

}

int termination\_trans()

{

    unsigned char aux;

    unsigned char trama[5];

    int state = 0;

    int res = 0;

    int total = 0;

    trama[0] = FLAG;

    trama[1] = A\_TRANS;

    trama[2] = DISC;

    trama[3] = (A\_TRANS^DISC);

    trama[4] = FLAG;

    (void) signal(SIGALRM, control\_alarm);

    while(tentat < NUMTRIES && state != 6)

    {

        if(STOP == FALSE){

            alarm(TIMEOUT);

            STOP = TRUE;

            res = write(fd, trama, 5);

            printf("DISC: 0x%X | 0x%X | 0x%X | 0x%X | 0x%X\n", trama[0], trama[1], trama[2], trama[3], trama[4]);

        }

        if(state != 5){

            res = read(fd, &aux, 1);

            total += res;

        }

        switch(state){

            case 0:

            if(aux == FLAG) state = 1;

            else total = 0;

            break;

            case 1:

            if(aux == A\_REC) state = 2;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 2:

            if(aux == DISC) state = 3;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 3:

            if(aux == (A\_REC^DISC)) state = 4;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 4:

            if(aux == FLAG) state = 5;

            else{

                state = 0;

                total = 0;

            }

            break;

            case 5:

            trama[2] = UA;

            trama[3] = (A\_REC^UA);

            res = write(fd, trama, 5);

            printf("UA: 0x%X | 0x%X | 0x%X | 0x%X | 0x%X\n", trama[0], trama[1], trama[2], trama[3], trama[4]);

            if(res == 5) state = 6;

            break;

        }

    }

    if(tentat == NUMTRIES) return -1;

    (void) signal(SIGALRM, SIG\_IGN);

    STOP = FALSE;

    tentat = 0;

    return 1;

}

int establishment\_rec()

{

    int state = 0;

    int total = 0;

    int res = 0;

    int count = 1;

    unsigned char inicio[5];

    unsigned char aux;

    inicio[0] = FLAG;

    inicio[1] = A\_REC;

    inicio[2] = UA;

    inicio[3] = (A\_REC^UA);

    inicio[4] = FLAG;

    while(state != 6)

    {

        if(state != 5){

            res = read(fd, &aux, 1);

            if(res == 0 && count != NUMTRIES) count++;

            else if(res == 0 && count == NUMTRIES) return -1;

            total += res;

        }

        switch(state)

        {

            case 0:

            if(aux == FLAG) state = 1;

            else total = 0;

            break;

            case 1:

            if(aux == A\_TRANS) state = 2;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 2:

            if(aux == SET) state = 3;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 3:

            if(aux == (A\_TRANS^SET)) state = 4;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 4:

            if(aux == FLAG){

                state = 5;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 5:

            res = write(fd, inicio, 5);

            printf("UA: 0x%X | 0x%X | 0x%X | 0x%X | 0x%X\n", inicio[0], inicio[1], inicio[2], inicio[3], inicio[4]);

            if(res == 5) state = 6;

            break;

        }

    }

    if(state == 6) return 1;

    else return -1;

}

unsigned char confirmationcheck()

{

    if(BCC2\_inicial == BCC2\_final){

        if(Nr\_rec == 1) return 0x21;

        else return 0x01;

    }

    else{

        if(Nr\_rec == 1) return 0x25;

        else return 0x05;

    }

}

int receiver\_information\_read(char\* packet)

{

    int state = 0;

    int j = 0;

    int res = 0;

    int total = 0;

    int count = 1;

    unsigned char aux;

    BCC2\_final = 0x00;

    BCC2\_inicial = 0x00;

    while(state != 5)

    {

        res = read(fd, &aux, 1);

        if(res == 0 && count != NUMTRIES) count++;

        else if(res == 0 && count == NUMTRIES) return 0;

        total += res;

        switch(state){

            case 0:

            if(aux == FLAG) state = 1;

            else total = 0;

            break;

            case 1:

            if(aux == A\_TRANS) state = 2;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 2:

            if(aux == 0x00 || aux == 0x02) state = 3;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 3:

            if(aux == 0x1){

                state = 4;

                packet[j] = aux;

                j++;

            }

            else if(aux == 0x0){

                state = 5;

                packet[j] = aux;

                j++;

            }

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 4:

            if(aux == ESC) status = 1;

            else if(aux == 0x5D && status == 1){

                status = 0;

                packet[j] = ESC;

                j++;

            }

            else if(aux == 0x5E && status == 1){

                status = 0;

                packet[j] = FLAG;

                j++;

            }

            else if(aux == FLAG) state = 5;

            else{

                packet[j] = aux;

                j++;

            }

            break;

        }

        if(state == 5){

            if(packet[0] == 0x1){

                BCC2\_inicial = packet[j-1];

                j--;

                for(int i = 0; i < j; i++){

                    BCC2\_final ^= packet[i];

                }

            }

            else break;

        }

    }

    if(BCC2\_inicial == BCC2\_final) return j;

    else return -1;

}

int receiver\_information\_write(char\* packet)

{

    unsigned char trama[5];

    int total;

    trama[0] = FLAG;

    trama[1] = A\_REC;

    trama[2] = confirmationcheck(packet);

    trama[3] = (trama[1]^trama[2]);

    trama[4] = FLAG;

    total = write(fd, trama, 5);

    if(total == 5) return 1;

    else return 0;

}

int termination\_rec()

{

    unsigned char aux;

    unsigned char trama[5];

    int state = 0;

    int res, total;

    int count = 1;

    trama[0] = FLAG;

    trama[1] = A\_REC;

    trama[2] = DISC;

    trama[3] = (A\_REC^DISC);

    trama[4] = FLAG;

    while(state != 11){

        if(state != 5){

            res = read(fd, &aux, 1);

            if(res == 0 && count != NUMTRIES) count++;

            else if(res == 0 && count == NUMTRIES) return -1;

            total += res;

        }

        switch(state){

            case 0:

            if(aux == FLAG) state = 1;

            else total = 0;

            break;

            case 1:

            if(aux == A\_TRANS) state = 2;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 2:

            if(aux == DISC) state = 3;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 3:

            if(aux == (A\_TRANS^DISC)) state = 4;

            else if(aux == FLAG){

                state = 1;

                total = 1;

            }

            else{

                state = 0;

                total = 0;

            }

            break;

            case 4:

            if(aux == FLAG) state = 5;

            else{

                state = 0;

                total = 0;

            }

            break;

            case 5:

            total = 0;

            res = write(fd, trama, 5);

            printf("DISC: 0x%X | 0x%X | 0x%X | 0x%X | 0x%X\n", trama[0], trama[1], trama[2], trama[3], trama[4]);

            if(res == 5) state = 6;

            break;

            case 6:

            if(aux == FLAG) state = 7;

            else total = 0;

            break;

            case 7:

            if(aux == A\_TRANS) state = 8;

            else if(aux == FLAG){

                state = 7;

                total = 1;

            }

            else{

                state = 6;

                total = 0;

            }

            break;

            case 8:

            if(aux == UA) state = 9;

            else if(aux == FLAG){

                state = 7;

                total = 1;

            }

            else{

                state = 6;

                total = 0;

            }

            break;

            case 9:

            if(aux == (A\_REC^UA)) state = 10;

            else if(aux == FLAG){

                state = 7;

                total = 1;

            }

            else{

                state = 6;

                total = 0;

            }

            break;

            case 10:

            if(aux == FLAG) state = 11;

            else{

                state = 6;

                total = 0;

            }

            break;

        }

    }

    return 1;

}

int llwrite(char\* buf, int bufSize)

{

    int totalwr = 0;

    int totalread = 0;

    if(buf == NULL || bufSize > MAX\_PAYLOAD\_SIZE) return -1;

    (void) signal(SIGALRM, control\_alarm);

    if(STOP == FALSE){

        alarm(TIMEOUT);

        STOP = TRUE;

    }

    totalwr = transmitter\_information\_write(buf, bufSize);

    //printf("Transmitter write ok: (totalwr) %d\n", totalwr);

    TOTALWRITE\_TRANS += totalwr;

    if(STOP == FALSE){

        if(tentat == NUMTRIES){

            resent\_write++;

            //llclose(TRUE);

            return -1;

        }

        else{

            resent\_write++;

            tentatREJ = 0;

            //sleep(1);

            return llwrite(buf, bufSize);

        }

    }

    else if(totalwr == 0){

        (void) signal(SIGALRM, SIG\_IGN);

        STOP = FALSE;

        tentat = 0;

        return -1;

    }

    totalread = transmitter\_information\_read();

    //printf("Transmitter read ok: (totalread) %d\n", totalread);

    if(totalread > 0) TOTALREAD\_TRANS += totalread;

    if(STOP == FALSE && totalread == 0){

        if(tentat == NUMTRIES){

            resent\_write++;

            //llclose(TRUE);

            return -1;

        }

        else{

            resent\_write++;

            tentatREJ = 0;

            //sleep(1);

            return llwrite(buf, bufSize);

        }

    }

    //REJ retrans

    if(totalread == -1){

        rej\_count\_trans++;

        (void) signal(SIGALRM, SIG\_IGN);

        STOP = FALSE;

        tentatREJ++;

        if(tentatREJ == NUMTRIES){

            //llclose(TRUE);

            return -1;

        }

        else return llwrite(buf, bufSize);

    }

    if(duplicado == 1){

        dup\_count\_trans++;

        duplicado = 0;

    }

    if(Nr\_trans == 0){

        Nr\_anterior\_trans = 0;

        Nr\_trans = 1;

    }

    else{

        Nr\_anterior\_trans = 1;

        Nr\_trans = 0;

    }

    if(Ns\_trans == 0){

        Ns\_anterior\_trans = 0;

        Ns\_trans = 1;

    }

    else{

        Ns\_anterior\_trans = 1;

        Ns\_trans = 0;

    }

    rr\_count\_trans++;

    (void) signal(SIGALRM, SIG\_IGN);

    tentatREJ = 0;

    tentat = 0;

    STOP = FALSE;

    if(totalwr < 0) return 0;

    else return totalwr;

}

int llread(char\* packet)

{

    int totalread = 0;

    int totalwr = 0;

    if(packet == NULL) return -1;

    totalread = receiver\_information\_read(packet);

    //printf("Receiver read ok: (totalread) %d\n", totalread);

    if(totalread > 0) TOTALREAD\_REC += totalread;

    if(totalread == 0){

        printf("Receiver didn't read nothing, closing connection...\n");

        return -1;

    }

    //REJ

    else if(totalread == -1){

        printf("BCC2 not equal, frame rejected...\n");

        rej\_count\_rec++;

        tentatREJ++;

        if(tentatREJ == NUMTRIES){

            //llclose(TRUE);

            return -1;

        }

        else return llread(packet);

    }

    totalwr = receiver\_information\_write(packet);

    //printf("Receiver write ok: (totalwr) %d\n", totalwr);

    TOTALWRITE\_REC += totalwr;

    if(totalwr == 0){

        printf("Error responding to frame (rx)...\n");

        return -1;

    }

    if(Ns\_rec == 0){

        Ns\_anterior\_rec = 0;

        Ns\_rec = 1;

    }

    else{

        Ns\_anterior\_rec = 1;

        Ns\_rec = 0;

    }

    if(Nr\_rec == 0){

        Nr\_anterior\_rec = 0;

        Nr\_rec = 1;

    }

    else{

        Nr\_anterior\_rec = 1;

        Nr\_rec = 0;

    }

    rr\_count\_rec++;

    tentatREJ = 0;

    if(totalread < 0) return -1;

    else return totalread;

}

int llclose(int showStatistics)

{

    int check\_tx, check\_rx;

    if(tx == 1) check\_tx = termination\_trans();

    else if(rx == 1) check\_rx = termination\_rec();

    if(check\_tx == -1) printf("Error closing the connection (tx)\n");

    else if(check\_rx == -1) printf("Error closing the connection (rx)\n");

    if(showStatistics == TRUE && tx == 1){

        printf("STATISTICS OF TRANSMITTER:\n");

        printf("Timeout defined: %d seconds\n", TIMEOUT);

        printf("Number of tries defined before closing: %d tries\n", NUMTRIES);

        printf("Number of bytes sent : %d bytes\n", TOTALWRITE\_TRANS);

        printf("Number of frames confirmed: %d frames\n", rr\_count\_trans);

        printf("Number of frames rejected: %d frames\n", rej\_count\_trans);

        printf("NUmber of frames duplicated: %d frames\n", dup\_count\_trans);

        printf("Number of frames retransmitted: %d frames\n", resent\_write);

    }

    else if(showStatistics == TRUE && rx == 1){

        printf("STATISTICS OF RECEIVER:\n");

        printf("Timeout defined: %d seconds\n", TIMEOUT);

        printf("Number of tries defined before closing: %d tries\n", NUMTRIES);

        printf("Number of bytes received: %d bytes\n", TOTALREAD\_REC);

        printf("Number of frames confirmed: %d frames\n", rr\_count\_rec);

        printf("Number of frames rejected: %d frames\n", rej\_count\_rec);

    }

    //REMINDER: final nr of bytes is different from the size of the picture because the protocol adds one byte to each frame transmitted

    if(tcsetattr(fd,TCSANOW,&oldtio) == -1){

        perror("tcsetattr");

        exit(-1);

    }

    close(fd);

    return 1;

}

constants.h

//constantes hexadecimais

#include <termios.h>

#ifndef CONSTANTS\_H

#define CONSTANTS\_H

#define FLAG 0x7E

#define A\_TRANS 0x03

#define A\_REC 0x01

//C pode ser 0x02 ou 0x00 (information frame)

#define ESC 0x7D

#define SET 0x03

#define DISC 0x0B

#define UA 0x07

// Opens a conection using the "port" parameters defined in struct linkLayer, returns "-1" on error and "1" on sucess

speed\_t get\_baud(int baud);

int establishment\_trans(); //returns 1 if everything goes well, -1 if error

int establishment\_rec(); //returns 1 if everything goes well, -1 if error

unsigned char informationcheck(); //returns the control flag for the transmitter of the information packet

unsigned char confirmationcheck(); //returns the control flag for the receiver of the information packet

int transmitter\_information\_write(char\* buf, int bufSize); //stuffing included, returns size of buffer if everything goes ok, if error returns 0

int transmitter\_information\_read(); //returns 1 if everything is ok, 0 if REJ was sent and -1 if timeout was activated during the read

int receiver\_information\_read(); //returns size of packet if everything is ok, 0 if REJ was sent and -1 if timeout was activated during the read

int receiver\_information\_write(char\* packet); //returns 1, if error returns 0

int termination\_trans(); //returns 1 if everything goes well, -1 if error

int termination\_rec(); //returns 1 if everything goes well, -1 if error

#endif

linklayer.h

#ifndef LINKLAYER

#define LINKLAYER

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <termios.h>

#include <stdio.h>

#include <signal.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

typedef struct linkLayer{

    char serialPort[50];

    int role; //defines the role of the program: 0==Transmitter, 1=Receiver

    int baudRate;

    int numTries;

    int timeOut;

} linkLayer;

//ROLE

#define NOT\_DEFINED -1

#define TRANSMITTER 0

#define RECEIVER 1

//SIZE of maximum acceptable payload; maximum number of bytes that application layer should send to link layer

#define MAX\_PAYLOAD\_SIZE 1000

//CONNECTION deafault values

#define BAUDRATE\_DEFAULT B38400

#define MAX\_RETRANSMISSIONS\_DEFAULT 3

#define TIMEOUT\_DEFAULT 4

#define \_POSIX\_SOURCE 1 /\* POSIX compliant source \*/

//MISC

#define FALSE 0

#define TRUE 1

// Opens a conection using the "port" parameters defined in struct linkLayer, returns "-1" on error and "1" on sucess

int llopen(linkLayer connectionParameters);

// Sends data in buf with size bufSize

int llwrite(char\* buf, int bufSize);

// Receive data in packet

int llread(char\* packet);

// Closes previously opened connection; if showStatistics==TRUE, link layer should print statistics in the console on close

int llclose(int showStatistics);

#endif

main.c

#include "linklayer.h"

#include <string.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

/\*

 \* $1 /dev/ttySxx

 \* $2 tx | rx

 \* $3 filename

 \*/

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

    if (argc < 4)

    {

        printf("usage: progname /dev/ttySxx tx|rx filename\n");

        exit(1);

    }

    printf("%s %s %s\n", argv[1], argv[2], argv[3]);

    fflush(stdout);

        if (strcmp(argv[2], "tx") == 0)

    {

        // \*\*\*\*\*\*\*\*\*\*\*

        // tx mode

        printf("tx mode\n");

        // open connection

        struct linkLayer ll;

        sprintf(ll.serialPort, "%s", argv[1]);

        ll.role = 0;

        ll.baudRate = 9600;

        ll.numTries = 3;

        ll.timeOut = 3;

            if(llopen(ll)==-1) {

                fprintf(stderr, "Could not initialize link layer connection\n");

                exit(1);

        }

        printf("connection opened\n");

        fflush(stdout);

        fflush(stderr);

        // open file to read

            char \*file\_path = argv[3];

            int file\_desc = open(file\_path, O\_RDONLY);

            if(file\_desc < 0) {

                fprintf(stderr, "Error opening file: %s\n", file\_path);

                exit(1);

            }

        // cycle through

        const int buf\_size = MAX\_PAYLOAD\_SIZE-1;

        unsigned char buffer[buf\_size+1];

        int write\_result = 0;

        int bytes\_read = 1;

        while (bytes\_read > 0)

        {

            bytes\_read = read(file\_desc, buffer+1, buf\_size);

                        if(bytes\_read < 0) {

                                fprintf(stderr, "Error receiving from link layer\n");

                                break;

                        }

            else if (bytes\_read > 0) {

                // continue sending data

                buffer[0] = 1;

                write\_result = llwrite(buffer, bytes\_read+1);

                    if(write\_result < 0) {

                            fprintf(stderr, "Error sending data to link layer\n");

                            break;

                    }

                            printf("read from file -> write to link layer, %d\n", bytes\_read);

            }

                        else if (bytes\_read == 0) {

                                // stop receiver

                                buffer[0] = 0;

                                llwrite(buffer, 1);

                                printf("App layer: done reading and sending file\n");

                                break;

                        }

            sleep(1);

        }

            // close connection

        llclose(1);

            close(file\_desc);

            return 0;

    }

    else

    {

            // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

            // rx mode

        printf("rx mode\n");

                struct linkLayer ll;

                sprintf(ll.serialPort, "%s", argv[1]);

                ll.role = 1;

                ll.baudRate = 9600;

                ll.numTries = 3;

                ll.timeOut = 3;

                if(llopen(ll)==-1) {

                        fprintf(stderr, "Could not initialize link layer connection\n");

                        exit(1);

                }

                char \*file\_path = argv[3];

                int file\_desc = open(file\_path, O\_RDWR|O\_CREAT, S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IROTH);

        if(file\_desc < 0) {

                        fprintf(stderr, "Error opening file: %s\n", file\_path);

                        exit(1);

                }

        int bytes\_read = 0;

        int write\_result = 0;

                const int buf\_size = MAX\_PAYLOAD\_SIZE;

                unsigned char buffer[buf\_size];

                int total\_bytes = 0;

        while (bytes\_read >= 0)

        {

                bytes\_read = llread(buffer);

                if(bytes\_read < 0) {

                        fprintf(stderr, "Error receiving from link layer\n");

                break;

            }

            else if (bytes\_read > 0) {

                if (buffer[0] == 1) {

                    write\_result = write(file\_desc, buffer+1, bytes\_read-1);

                                if(write\_result < 0) {

                                    fprintf(stderr, "Error writing to file\n");

                                        break;

                                }

                                total\_bytes = total\_bytes + write\_result;

                    printf("read from file -> write to link layer, %d %d %d\n", bytes\_read, write\_result, total\_bytes);

                }

                else if (buffer[0] == 0) {

                    printf("App layer: done receiving file\n");

                    break;

                }

            }

                }

                llclose(1);

                close(file\_desc);

                return 0;

    }

}