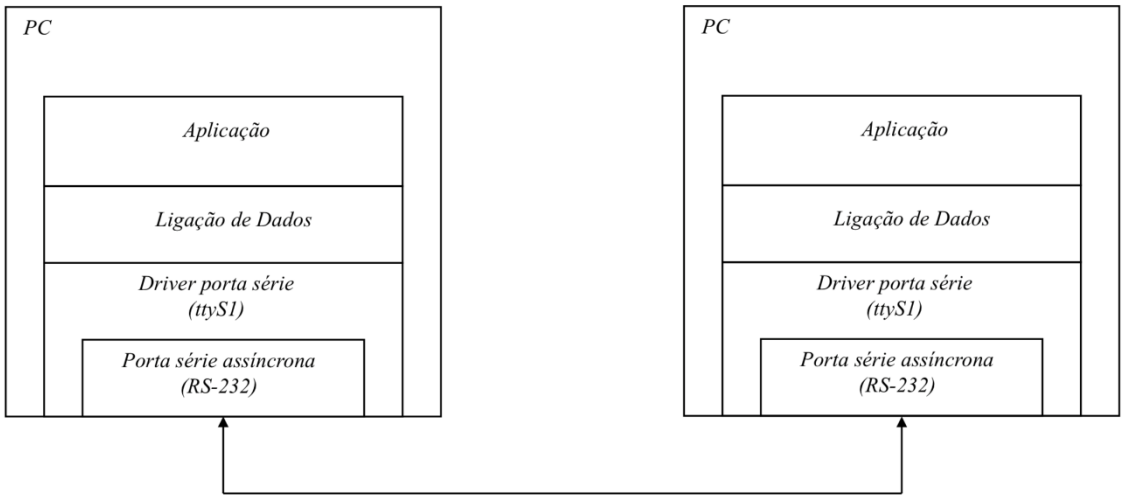
U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Relatório do 1º Trabalho Laboratorial

Protocolo de Ligação de Dados



Irene Amaral, up201506644
Rodrigo Caldas, up201708987
Tiago Ribeiro, up20170988

Índice

Introdução

Este trabalho teve como objetivo a transferência de ficheiros através de uma porta série. Primeiramente, elaborou-se uma camada de ligação de dados e, de seguida, desenvolveu-se uma camada de aplicação que permitisse a transferência destes.

Após ter sido implementado e devidamente demostrado na aula laboratorial da unidade curricular de Redes de Computadores, o primeiro projeto concluiu com sucesso as três avaliações primárias que consistiam no envio de um ficheiro pela porta série, seguido do envio do mesmo ficheiro pela porta, mas desta vez com interrupção da porta a meio da transmissão e por fim, é novamente enviado o ficheiro mas com existência de ruído na porta série.

Biblioteca de Funções

```
typedef struct {
        unsigned char* nome;
        int tamanho;
} detalhes;  (parâmetros nome e tamanho do ficheiro)

typedef struct {
        unsigned char T; //Type
        unsigned char L; //Length
        unsigned char *V; //Value
} parametros; (parâmetros do ficheiro no formato TVL – type, length, value)
```

```
typedef struct { //pacote de dados
        unsigned char N;
        unsigned char L1;
        unsigned char L2;
        unsigned char *dados;
} data;  (parâmetros do pacote de Dados)
```

int ler_trama(int port, unsigned char *trama) lê a trama da porta.

void trama_inicial(unsigned char* trama, unsigned char A, unsigned char C) forma a trama inicial.

int tramatipo_I(unsigned char* trama, unsigned char* buff, int length, int sendNumber) onde é feito o byte stuffing.

int construir_pacoteTVL(unsigned char C, unsigned char* pacote, parametros* tvl) preenche pacote com o campo C, os TLVs de tvl e retorna o seu tamanho como inteiro.

int construir_pacoteDados(unsigned char* buff, unsigned char* pacote, int tamanho_pacote, int* seq_num) coloca os dados de buff, com tamanho tamanho_pacote, num pacote pacote com um número de sequência seq_num passado por referência, e incrementa-o.

void pacote_tvl(unsigned char *pacote, parametros *tvl) preenche os TLVs de tvl com os dados em pacote.

void pacotedados(unsigned char *pacote, data *pacote_data); alterar o pacote para uma estrutura pacote_data, do tipo data.

int porta(char *port, struct termios *oldtio) set da porta e retorna o seu apontador.

int getFileLength(int fd) retorna o tamanho do ficheiro.

Camada de Ligação de Dados

A camada de ligação de dados tem como objetivo estabelecer e terminar a conexão - **llopen()** e **llclose()** -, escrever e/ou ler os dados da porta série, fazer o tratamento de erros e é também responsável por fazer o byte *stuffing/destuffing* de pacotes – **llread()** e **llwrite()**.

Para auxílio destas funções, foi criada a função **ler_trama()** que lê através da porta e guarda numa trama o que lê.

1. llopen(int port, int Tx_Rx)

Esta função é chamada na camada de Aplicação, que terá como argumentos a porta série e o Transmissor ou Recetor. Se o segundo argumento for o Transmissor, a função constrói a trama SET, e envia-a para a porta. Fica assim a aguardar pela resposta UA. Caso não receba, a função tenta enviar a trama novamente. Ao fim de 3 tentativas se continuar sem receber qualquer resposta por parte do Recetor, a função retorna um erro e não se estabelece conexão.

Se o argumento for o Recetor, a função fica à espera de receber o SET, e se o receber envia uma trama UA e a ligação é estabelecida. Se não se verificar isto, voltamos a ter as 3 tentativas de espera para receber o SET e retornando erro em caso de ultrapassar o número máximo de tentativas, ou seja, 3.

2. llread(int port, unsigned char* buffer)

Inicialmente a função aguarda a receção de uma trama pela porta. Se não receber é iniciado um time-out de 3 tentativas (no máximo) de receção e caso não haja sucesso é enviado um REJ.

Se houver sucesso na receção da trama, é calculado o BCC1 e confirmado com base na seguinte condição: BCC1=A^C. Se não estiver correto é novamente enviado um REJ. Se estiver correto, passa ao próximo estado onde é verificado se equivale a uma trama de DISC. Se efetivamente for recebido um DISC é chamada a função llclose() e termina-se a conexão. Senão procede-se ao byte destuffing do pacote recebido e do BBC2. Após serem verificados se estiver tudo bem, envia um RR senão envia REJ.

Por fim, se for recebida uma trama duplicada, por exemplo, devido a uma trama RR perdida, responde com uma trama RR, sendo a trama descartada.

3. llwrite(int port, unsigned char* buffer, int length)

Esta função recebe um pacote (de controlo ou de dados) e constrói uma trama de Informação envia-o para a porta série. Fica então à espera de uma resposta do Recetor e se receber uma trama RR altera o seu Nr (troca 0 por 1 ou 1 por 0, dependendo do caso) e retorna o número de caracteres escritos na porta. Se não receber um RR mas sim um REJ é enviada novamente a trama I.

4. llclose(int port, int Tx_Rx)

Esta função é chamada na camada de Aplicação e será o contrário da llopen() pois neste caso irá terminar a conexão. Se o segundo argumento for o Transmissor, a função constrói a trama DISC, e envia-a para a porta. Fica assim a aguardar pela resposta DISC. Caso não receba, a função tenta enviar a trama novamente. Ao fim de 3 tentativas se continuar sem receber qualquer resposta por parte do Recetor, a função retorna um erro e não se estabelece conexão. Se for recebido, envia UA e está terminada a conexão com sucesso.

Se o argumento for o Recetor, a função envia o DISC e fica à espera de receber um UA e está assim terminada a conexão com sucesso. Se não se verificar isto, voltamos a ter as 3

tentativas de espera para receber o UA e retornando erro em caso de ultrapassar o número máximo de tentativas, ou seja, 3.

Camada de Aplicação

Na camada de aplicação foram desenvolvidas duas funções, transmissor() e recetor() que serão posteriormente chamadas na função main() da camada de aplicação.

1. transmissor(int fd_port, char *path)

Esta função é responsável por ler os dados do ficheiro que se pretende enviar para a camada de ligação de dados e por transmiti-lo se seguida. Cria um pacote de controlo, neste caso um o primeiro é um pacote Start (C=2) e codifica-o no formato TLV (Type, Length, Value). Para cada parâmetro do pacote de controlo, é indicado o tipo de parâmetro (T), seguido do tamanho desse parâmetro em bytes (L) e o valor do parâmetro (V). No protocolo são apenas usados dois parâmetros: tamanho e nome.

Estando por fim construído o pacote Start é enviado pela função llwrite().

De seguida, lê os dados do ficheiro e constrói os pacotes de dados que são enviados pela função llwrite(). Os pacotes de dados são formados pelo campo de controlo (C=1), um byte N – número de sequência (módulo 255), seguido de dois bytes, L2 e L1, que indicam o número de bytes do campo de dados e por fim o campo de dados do ficheiro.

O número de bytes no campo de dados obedece à seguinte condição: K=256*L2+L1, sendo neste caso o número máximo dado por,

256 × (28 − 1) + (28 − 1) = 65535. Terá ainda ser adicionado 4 bytes devido à existência dos campos C, N, L2 e L1.

Concluído o envio dos pacotes de dados (no caso do pinguim.gif é apenas enviado num único pacote), é contruído novamente um pacote de controlo, neste caso um pacote END (C=3) e enviado.

Por fim é chamada a função llclose() e termina a conexão com sucesso.

2. recetor(int fd_port)

Esta função lê os dados da camada de ligação e escreve-os no novo ficheiro criado. Verifica para cada pacote (Start, Dados e End), o campo de controlo, o número de sequência N, o tamanho de L2 e L1 e compara com tamanho obtido no buffer. Lê também o pacote de dados com auxílio da função llread() onde faz o byte destuffing. Se for lido um DISC a função llread() interpreta como o fim da conexão e, portanto, é chamada a função llclose() e termina assim a conexão com sucesso.

Validação

O projeto cumpre com sucesso os testes a que foi sujeito, tais como:

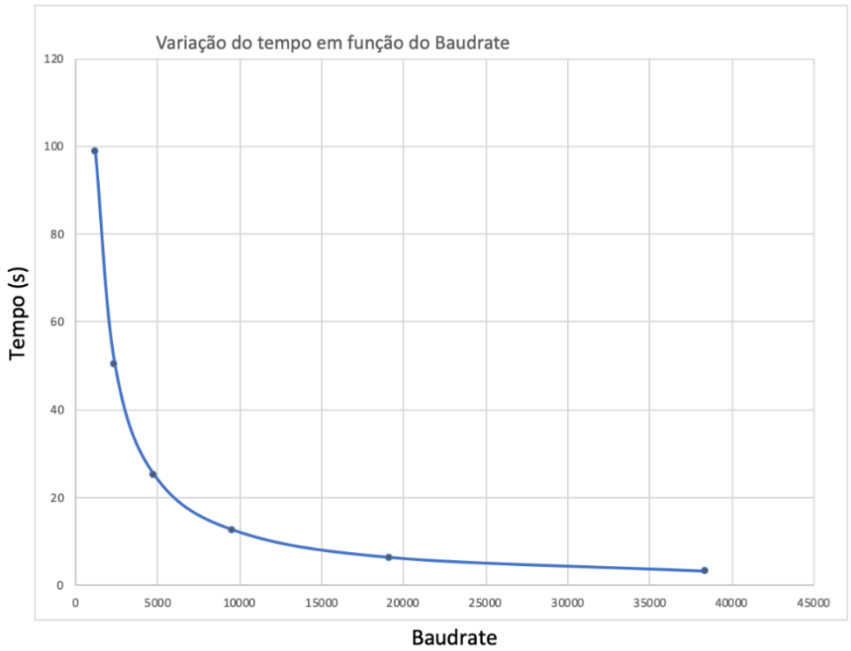
- Enviar um ficheiro pela porta série;
- Enviar um ficheiro interrompendo a transmissão da porta série;
- Enviar um ficheiro gerando ruído na porta série;

Eficiência do protocolo de ligação de dados

Nota: para os casos de estudo estatístico foi enviado o ficheiro penguin.gif de tamanho 10 968 bytes.

1. Caso 1: Variação do *Baudrate*

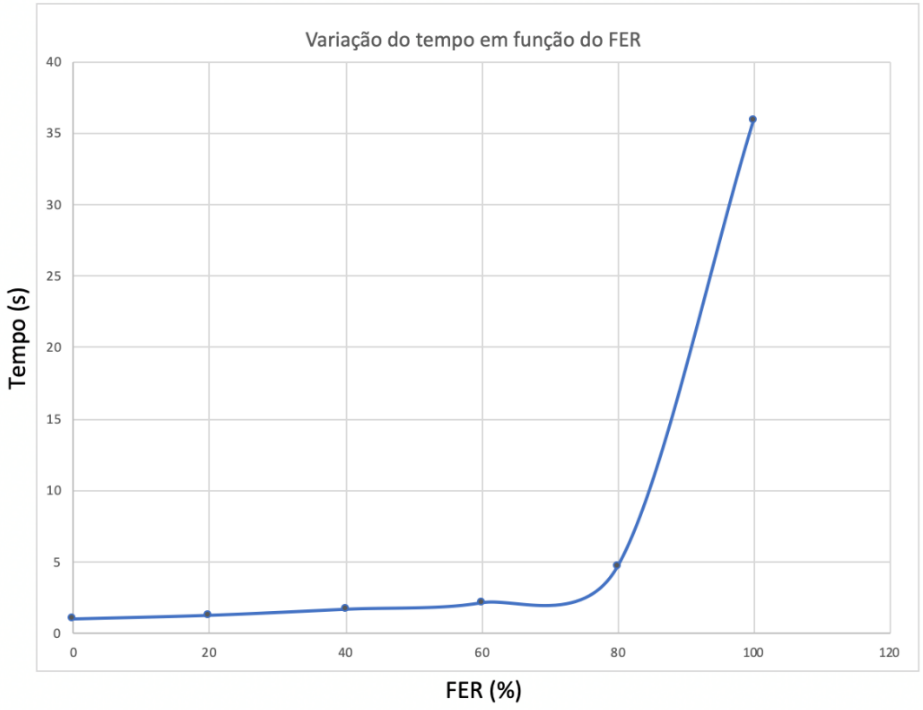
VARIAR O BAUDRATE		
Baudrate	tempo (s)	S=R/C (%)
1200	98,6312	98,26
2400	50,0324	96,85
4800	25,0019	96,83
9600	12,4846	96,81
19200	6,2271	96,77
38400	3,0993	96,65
57600	2,0513	96,78
115200	1,0114	96,42



Verifica-se que quanto menor o *baudrate*, maior é o tempo da transferência do ficheiro. A eficiência é maior para valores mais pequenos de *baudrate*.

2. Caso 2: Variação do FER

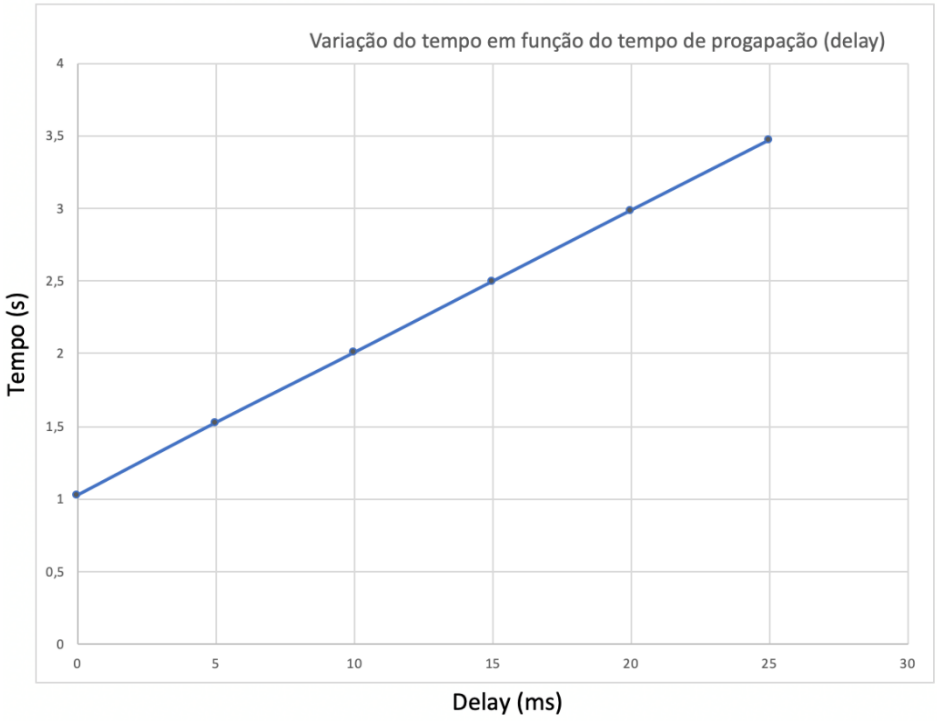
VARIAR FER (FRAME ERROR RATE)	
%	tempo (s)
0	1,0252
20	1,2818
40	1,7022
60	2,1567
80	4,7072
100	35,8872



Variando o *Frame Error Ratio*, para 6 amostras, concluímos que quanto menor é o FER menor é o tempo necessário para transmitir o ficheiro, tal como era de esperar.

3. Caso 3: Variação do tempo de propagação

VARIAR O TEMPO DE PROPAGAÇÃO	
Delay (ms)	tempo (s)
0	1,0252
5	1,5235
10	2,0048
15	2,494
20	2,9816
25	3,4648



Verifica-se que quanto maior o tempo de propagação (delay), maior será o tempo de transferência do ficheiro, como era expectável.

Conclusão

No geral, a implementação do protocolo foi um atingida com sucesso. Este é assim capaz de enviar um ficheiro através de uma porta série mesmo com as dificuldades e erros impostos propositadamente na sua transmissão como foi avaliado pelo docente na aula laboratorial.

Os valores elevados obtidos para a eficiência da transmissão comprovam de igual forma o sucesso do protocolo.

Desta forma, o projeto foi uma mais valia na consolidação dos fundamentos teóricos lecionados ao longo das aulas.

Anexos

1. Camada_de_aplicação.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <signal.h>
#include <inttypes.h>
#include <time.h>
#include "biblio.h"
#include "camada_de_ligacao.h"

#define erro -1
#define erro_duplicada -2
int trama_cont;      //nr tramas enviados

int transmissor(int fd_port, char *path) {

    unsigned char buff[TAM_BUF], pacote[TAM_BUF];
    int abrir_path, aux=0;
    int  estado=0, ciclo=0, seq_num=0, i=0;
    int count_bytes2=0, ler=0;
    int count_bytes=0;
    int trama_cont=0;
    parametros tvl[tamanho_trama];
    detalhes ficheiro;
    struct stat fileStat;

    while(!ciclo) {
        switch(estado) {
            case 0:
                if(stat(path, &fileStat) < 0) { //saber info sobre um ficheiro at
raves do seu path
                    printf("Erro no stat()");
                    return -1;
```

```
                }

                if((ficheiro.nome = (unsigned char *) malloc(strlen(path)+1)) ==
NULL) {
                    printf("erro no malloc para o nome do ficheiro");
                    return -1;
                }

                strcpy((char *) ficheiro.nome, path);

                abrir_path = open(path, O_RDONLY);
                if(abrir_path<0){
                    printf("ERRO NO ABRIR_PATH");
                    return -1;
                }

                printf("\nABRINDO O LLOPEN() NO TX\n");

                 aux = llopen(fd_port, TX);

                if(aux == -5) { //verificação de desconexão
                    estado=0;
                    break;
                }
                if(aux < 0) {
                    printf("Erro no llopen()");
                    return -1;
                }
                if(aux == 0) {
                    printf( "\n\nLLOPEN FUNCIONOU COM SUCESSO NO TX.\n\n");
                }

                    if((ficheiro.tamanho = getFileLength(abrir_path)) < 0) {
                    return -1;
                }


            case 1:
                //TVL É PARA OS PACOTES START E END
                //PACOTE START 2
                tvl[0].T = 0; //Tamanho
                tvl[0].L = sizeof(ficheiro.tamanho);
                tvl[0].V = (unsigned char *) malloc(tvl[0].L);

                tvl[1].T = 1; //Nome
                tvl[1].L = strlen((char *) ficheiro.nome) + 1;
```

```
                tvl[1].V = (unsigned char *) malloc((int) tvl[1].L);
                memcpy(tvl[1].V, ficheiro.nome, (int) tvl[1].L);

                aux = construir_pacoteTVL(C_START, pacote, tvl);
                if(aux < 1){
                    printf("\n\nerro no construir_pacoteTVL\n\n");
                    return -1;
                }

                aux = llwrite(fd_port, pacote, aux);

                if(aux == -5) { // verificação de desconexão
                    estado=0;
                    break;
                }

                if(aux < 0) {
                    printf("\n\nerro no llwrite no pacote start\n\n");
                    return -1;
                }
                estado = 2;
                break;

            case 2: //PACOTE DE DADOS 1
                while(count_bytes2 < ficheiro.tamanho) {

                    if(count_bytes2 + TAM_BUF-4 < ficheiro.tamanho) {
                        ler = TAM_BUF-4;
                        count_bytes2 = count_bytes2+ler;
                    } else {
                        //ler o ultimo byte
                        ler = ficheiro.tamanho - count_bytes2;
                        count_bytes2 = count_bytes2+ler;
                    }
                    if((aux = read(abrir_path, buff, ler)) < 0) {
                        printf("erro no read()");
                        return -1;
                    }

                    if((aux = construir_pacoteDados(buff, pacote, aux, &seq_num))
 < 0) { //coloca os dados do buff num pacote
                        printf("\n\nerro no construir_pacoteDados\n\n");
                        return -1;
                    }
                    aux = llwrite(fd_port, pacote, ler+4); //retorna número de ca
racteres escritos do pacote para a porta
```

```
            if(aux == -5) { // verificação de desconecção
                estado=0;
                break;
            }
            if(aux < 0) {
                printf("erro llwrite() pacote dados");
                return -1;
            } else {
                count_bytes =count_bytes + aux;
            }
            //limpar o buffer
            for(int clr = 0; clr < TAM_BUF; clr++) {
                buff[clr] = 0;
            }
        }
        estado=3;
    break;

    case 3:
    //PACOTE END 3
        aux = construir_pacoteTVL(C_END, pacote, tvl);
        if(aux < 1){
            printf("erro no construir_pacoteTVL");
            return -1;
        }

        aux = llwrite(fd_port, pacote, aux);
        if(aux == -5) { // verifivação de desconexão
            estado=0;
            break;
        }
        if(aux < 0) {
            printf("erro no llwrite pacote end");
            return -1;
        }
        estado = 4;
        break;


    break;
    //close por parte do TX
    case 4:
        printf("\n\n ABRINDO O LLCLOSE NO TX.\n");

        aux = llclose(fd_port, TX);
        if(aux == -5) { // verifivação de desconexão
```

```
                    estado=0;
                    break;
                }

                if(aux < 0) {
                    printf("Erro no llclose() - transmissor");
                    return -1;
                }
                if(aux == 0) {
                    printf("\n\n LLCLOSE FUNCIONOU COM SUCESSO NO TX.\n\n");
                }

                ciclo = 1;
                break;

            default:
                break;
        }
    }

    return 0;
}

int recetor(int fd_port) {
    parametros tvl_start[tamanho_trama], tvl_end[tamanho_trama];
    data dados_app;
    detalhes start, end;
    unsigned char buff[TAM_BUF];
    unsigned char pacote[TAM_BUF-1];   //nao contem o campo C
    int output=0, aux=0, ciclo=0, estado=0, i=0, j=0, change=0, count_bytes=0, se
q_N=255;
    int ciclo_1=0, ciclo_2=0;

    printf( "\n\t ABRINDO LLOPEN NO RX\n\n");

    test: aux = llopen(fd_port, RX);
    if(aux == -5) { // verificação de desconexão
        estado=0;
        goto test;
    }
    if(aux < 0) {
        printf("Erro no llopen() - recetor");
        return -1;
    }
    if(aux == 0) {
        printf("\n\n LLOPEN FUNCIONOU COM SUCESSO NO RX.\n\n");
```

```
    }
    while (!ciclo_1) {
        switch (estado) {
            case 0:
            //LEITURA DO PACOTE
                if (!ciclo_2) {

                    aux = llread(fd_port, buff);
                    if(aux == -5) { // verificação de desconexão
                        estado=0;
                        goto test;
                        break;
                    }
                    if (aux == erro) { //erro =-
1 que é quando no llread lê um DISC (fim)

                        ciclo_1 = 1;

                        break;
                    } else if (aux < 0) {
                        printf("erro no llread() - recetor");
                        return -1;
                    }
                    ciclo_2 = 1;
                }
                if (ciclo_2) {
                    //Escrita do que leu do buffer no pacote (no buf é excluido o
 campo C)
                    for (i = 0; i < aux-1; i++) {
                        pacote[i] = buff[i+1];
                    }
                    //SE FOR Pacote de Controlo Start
                    if (buff[0] == C_START){
                        pacote_tvl(pacote, tvl_start);

                    } else if (buff[0] == C_DADOS) {
                        ciclo_2 = 1;
                        estado = 1;
                        break;
                    } else {
                        printf("\n\nErro no campo C_Dados\n");
                        return -1;
                    }
                    ciclo_2 = 0;
                }
```

```
                //O NOME QUE DEMOS TEM DE SE ALTERAR PARA start.file_name
                if((output = open("penguin.gif", O_CREAT | O_WRONLY, S_IROTH | S_
IWOTH | S_IXOTH | S_IXGRP | S_IWGRP | S_IRGRP | S_IXUSR | S_IWUSR | S_IRUSR)) < 0
) {
                    printf("Erro no open do ficheiro no recetor");
                    return -1;
                }

                break;


                case 1:
                //Pacote de dados
                if (!ciclo_2) {

                    aux = llread(fd_port, buff);
                    if(aux == -5) { // verificação de desconexão
                        estado=0;
                        goto test;
                        break;
                    }
                    if (aux == erro) { //Leu o DISC - Fim da leitura
                        ciclo_1 = 1;
                        break;
                    } else if (aux == erro_duplicada) {
                        break; //se receber frame repetida no llread retorna -
2 (trama duplicada); ignora a leitura do duplicado; volta a ler a proxima
                    } else if (aux < 0) {
                        printf("erro no llread()");
                        return -1;
                    }
                    ciclo_2 = 1;
                }

                if (ciclo_2) {
                    //Leitura do pacote de Dados
                    for (i = 0; i < aux-1; i++) {
                        pacote[i] = buff[i+1];
                    }
                    if (buff[0] == C_DADOS) { //RECEBEU FLAG DE DADOS (1) - JA LE
U DO BUFFER E GUARDOU NO PACOTE E VAI PREENCHER DEPOIS COM A FUNÇÃO EM BAIXO NO D
ADOS_APP

                        //o tamanho recebido do llread da porta e guardado no buf
f é diferente do esperado 256*L2 +L1)
```

```
                        if ((aux-4) != (256*(int)pacote[1] + (int)pacote[2])) {
                            printf("O tamanho máximo do buffer foi ultrapassado.
ERRO!!!");
                        }


                        pacotedados(pacote, &dados_app);  //alterar o pacote para
 uma estrutura do tipo data
                    } else if (buff[0] == C_END) {
                        ciclo_2 = 1;
                        estado = 2; //salta para o case 2 - pacote END
                        break;
                    }
                    //C - campo de controlo (valor: 1 - dados)
                    //N - número de sequência (módulo 255)
                    //L2 L1 -indica o número de octetos(K) do campo de dados
                    //P1 ... PK - campo de dados do pacote (K octetos)

                    if (((int)dados_app.N - seq_N == 1) || (seq_N - (int)dados_ap
p.N == 255)) {
                        seq_N = (int)dados_app.N; //nr de sequencia
                        if (seq_N == 256) {
                            seq_N = 0;
                        }
                    } else {
                        printf("\n\nErro no número de sequencia.\n");
                        return -1;
                    }
                    //write(int output, const void *buf(o que vai ser escrito), s
ize_t nbytes(256*L2 +L1));
                    if ((aux = write(output, dados_app.dados, 256*(int)dados_app.
L2 + (int)dados_app.L1)) < 0) {
                        printf("write()");
                        return -1;
                    }

                    for (i = 0; i < TAM_BUF; i++) {
                        buff[i] = 0;
                    }

                    ciclo_2 = 0;
                }
                break;

                case 2:
                //PACOTE END
```

```
            if (!ciclo_2) {
                aux = llread(fd_port, buff);
                if(aux == -5) { // verificação de desconexão
                    estado=0;
                    goto test;
                    break;
                }
                if (aux == erro) { //Leu o DISC - Fim da leitura
                    ciclo_1 = 1;
                    break;
                } else if (aux < 0) {
                    printf("erro no llread()");
                    return -1;
                }
                ciclo_2 = 1;
            }

            if (ciclo_2) {
                //Leitura do pacote END
                for (i = 0; i < aux-1; i++) {
                    pacote[i] = buff[i+1];
                }
                if (buff[0] == C_END){
                    pacote_tvl(pacote, tvl_end);
                }
                ciclo_2 = 0;
            }
            break;
    }
}

printf( "\n\n ABRINDO LLCLOSE NO RX.\n\n");

aux = llclose(fd_port, RX);
if(aux == -5) { //verificação de desconexão
        estado=0;
        goto test;
}
if(aux < 0) {
    printf("Erro no llclose()");
    return -1;
}
if(aux == 0) {
    printf("\n\n LLCLOSE FUNCIONOU COM SUCESSO NO RX.\n\n");
}
if(close(output) < 0) {
```

```
        printf("erro no close do ficheiro");
        return -1;
    }

    return 0;
}


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

    int fd_port;
    struct termios oldtio;
    char buf[100];

    if (argc < 2) {
        printf("\n\tex: nserial /dev/ttyS0\n");
        exit(-1);
    }

    if((fd_port = porta(argv[1], &oldtio)) < 0) {
        printf("Erro na porta");
        exit(-1);
    }

    printf("RX OU TX?\n\n");
    if(fgets(buf, sizeof(buf), stdin) == 0){
        exit(-1);
    }
    if((strncmp(buf, "TX", 2) == 0)){
        if(transmissor(fd_port, argv[2]) < 0) {
            printf("Erro no transmissor");
            exit(-1);
        }
    } else if((strncmp(buf, "RX", 2) == 0)) {
        if(recetor(fd_port) < 0) {
            printf("Erro no recetor");
            exit(-1);
        }
    } else {
        printf("\nNão foi inserido correntamente TX/RX.\n");
        exit(-1);
    }

    sleep(1);
    return 0;
```

```
}
```

2. Camada_de_ligação.c

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include "biblio.h"
#include "camada_de_ligacao.h"
#define erro -1

int N_trama_ler = 0;        //Numero da trama a ler
int N_trama_escrever  = 0;  //Numero da trama a escrever
int trama_cont=0;           //frames enviados

//LLOPEN PRONTO
int llopen(int port, int Tx_Rx) { // Cria a ligação entre os 2 pontos

    int estado = 0, aux = 0, ciclo1 = 0, time_out_cont = 0, aux_trama = 0;
    unsigned char SET[5];
    unsigned char UA[5];

    unsigned char trama_recebida[trama_length ];

    //TRAMAS DE CONTROLO SET E UA DO TIPO [F A C Bcc F]
    trama_inicial(SET, FR_A_TX, FR_C_SET);
    trama_inicial(UA, FR_A_TX, FR_C_UA);

    if(Tx_Rx == TX) { //TX=1
        while (!ciclo1) {
            switch (estado) {
                case 0:
                    //Inicio da conexão - envio do SET
                    tcflush(port, TCIOFLUSH);
                    aux = write(port, SET, 5);
                    if(aux < 0) {
                        printf("Erro na escrita do set");
```

```
                    return -1;
                }
                estado = 1;
                break;

            case 1:
                //Receber UA
                aux_trama = ler_trama(port, trama_recebida);
                if(aux_trama==-5) return -5; // cabo com má conexão
                if(aux_trama == erro) {
                    if(time_out_cont < 3) {
                        printf("NENHUM UA FOI RECEBIDO + 1 time_out\n");
                        sleep(1);
                        time_out_cont++;
                        estado = 0; //volta ao estado 0
                    } else {
                        printf("NENHUM UA FOI RECEBIDO - APÓS AS 3 TENTATIVAS
\n");
                        return -1; //ultrapassadas as 3 tentativas
                    }
                } else {
                    if(memcmp(trama_recebida, UA, 5) == 0) { //comparar os pr
imeiros 5 bits da trama e do UA (se =0 é igual)
                        printf("A conexão foi estabelecida.\n");
                        ciclo1 = 1;
                    } else {
                        estado = 0; //volta ao estado 0
                    }
                }
                break;

            default:
                break;
        }
    }

} else if(Tx_Rx == RX) { //RX=0
    while(!ciclo1) {
        switch(estado) {
            case 0:
            //RECEBER O SET
                aux_trama = ler_trama(port, trama_recebida);
                if(aux_trama==-5) return -5; // cabo com má conexão
                if(aux_trama == erro) {
                    if(time_out_cont < 3) {
                        printf("\nNada foi recebido no RX.\n");
```

```
                            sleep(1);
                            time_out_cont++;
                            estado = 0;
                        } else {
                            printf("Nenhum SET foi recebido após as 3 tentativas\
n");
                            return -1; //ultrapassadas as 3 tentativas
                        }
                    } else {

                        if(memcmp(trama_recebida, SET, 5) == 0) {
                            estado = 1;
                        }
                    }
                    break;

                case 1:

                    tcflush(port, TCIOFLUSH);
                    //ENVIO DA TRAMA UA
                    aux = write(port, UA, 5);
                    if(aux < 0) {
                        printf("Erro na escrita do UA");
                        return -1;
                    }
                    printf("\nA conexão foi estabelecida.\n");
                    ciclo1 = 1;
                    break;

                default:
                    break;
            }
        }
    }

    return 0;

}
//LLREAD PRONTO
int llread(int port, unsigned char *buffer) { // Vai ser retornado o comprimento
da trama sem os comprimentos dos cabeçalhos

  int estado = 0, time_out_cont = 0, i = 0, j = 0, ciclo1 = 0, aux_trama = 0;
    unsigned char RR[5], REJ[5], RR_PERDIDO[5], DISC[5];
    unsigned char trama_recebida[trama_length ]; //[F A C BCC F]=[0 1 2 3 4]
    unsigned char BCC2 = 0;
```

```
    //DISC
    trama_inicial(DISC, FR_A_TX, FR_C_DISC);

    while(!ciclo1) {
        switch(estado) {
            case 0:
                aux_trama = ler_trama(port, trama_recebida);
                if(aux_trama==-5) return -5; // cabo com má conecção
                if(aux_trama == erro) {
                    if(time_out_cont < 3) {
                        printf("Nada recebido + 1 time_out\n");
                        sleep(1);
                        time_out_cont++;
                        estado = 6;
                    } else {
                        printf("Nada recebido após as 3 tentativas\n");
                        return -3;
                    }
                } else {
                    if (trama_recebida[2] != FR_C_SET && trama_recebida[2] != FR_
C_UA && trama_recebida[2] != FR_C_DISC) {
                    }
                    estado = 1;
                }
                break;

            case 1:
            //Tramas de Supervisão(S) e Não Numeradas(U)
                    //BCC1=A^C
                if(trama_recebida[3] != (trama_recebida[1]^trama_recebida[2])) {

                    estado = 6;  //ENVIADO rej(rejeição/ACK NEGATIVO)
                    break;
                }
                estado = 2;
                break;

            case 2:
            //Ver se recebeu um disc
                if(memcmp(trama_recebida, DISC, 5) == 0) {
                    return -1;
                } else {
                    estado = 3;
                }
                break;
```

```
            case 3:
            //destuffing ; VER Transparency(guião ingles)
            //evitar o falso reconhecimento de uma flag no interior de uma trama
                for (i = 4; i < aux_trama - 1; i++) {
                    if((trama_recebida[i] == ESC) && ((trama_recebida[i+1] == FR_
F_AUX) || (trama_recebida[i+1] == ESC_AUX))) {
                        if(trama_recebida[i+1] == FR_F_AUX) {
                            trama_recebida[i] = FR_F;
                        } else if(trama_recebida[i+1] == ESC_AUX) {
                            trama_recebida[i] = ESC;
                        }
                        for (j = i + 1; j < aux_trama - 1; j++){
                            trama_recebida[j] = trama_recebida[j+1];
                        }
                        aux_trama--;
                    }
                }
                estado = 4;
                break;

            case 4:
                BCC2 = trama_recebida[4]; //Bcc2: Tramas de Informação (I) - guiã
o inglês COMPARAR SE É IGUAL AO CALCULO DO XOR
                for (i = 5; i < aux_trama - 2; i++) {
                    BCC2 = BCC2 ^ trama_recebida[i];
                }
                if (BCC2 != trama_recebida[aux_trama-2]) {
                    estado = 6; // SALTA PARA REJ
                    break;
                }
                estado = 5;
                break;

            case 5:
                if(trama_recebida[2] == FR_C_SEND0 && N_trama_ler == 0) {
                    N_trama_ler = 1;
                    trama_inicial(RR, FR_A_TX, FR_C_RR1);
                } else if (trama_recebida[2] == FR_C_SEND1 && N_trama_ler == 1) {
                    N_trama_ler = 0;
                    trama_inicial(RR, FR_A_TX, FR_C_RR0);
                } else {
                    estado = 7;
                    break;
                }
```

```
                for (i = 4, j = 0; i < aux_trama - 2; i++, j++) {
                    buffer[j] = trama_recebida[i];
                }
                //enviar RR - correu tudo bem
                tcflush(port, TCIOFLUSH);
                if (write(port, RR, 5) < 0) {
                    printf("Erro na escrita do RR");
                    return -3;
                }
                ciclo1 = 1;
                break;

            case 6:
            //envia rej
                if (aux_trama == erro) {
                    if(N_trama_ler == 0) {
                        trama_inicial(REJ, FR_A_TX, FR_C_REJ0);
                    } else if (N_trama_ler == 1) {
                        trama_inicial(REJ, FR_A_TX, FR_C_REJ1);
                    }
                } else {
                    if(trama_recebida[2] == FR_C_SEND0 && N_trama_ler == 0) {
                        trama_inicial(REJ, FR_A_TX, FR_C_REJ0);
                    } else if (trama_recebida[2] == FR_C_SEND1 && N_trama_ler ==
1) {
                        trama_inicial(REJ, FR_A_TX, FR_C_REJ1);
                    }
                }

                //ENVIAR REJ - REJEIÇÃO
                tcflush(port, TCIOFLUSH);
                if (write(port, REJ, 5) < 0) {
                    printf("Erro na escrita do REJ");
                    return -3;
                }
                estado = 0;
                break;

            case 7: //RR PERDIDO / TRAMA DUPLICADA
                if (trama_recebida[2] == FR_C_SEND0 && N_trama_ler == 1) {
                        trama_inicial(RR_PERDIDO, FR_A_TX, FR_C_RR1);
                } else if (trama_recebida[2] == FR_C_SEND1 && N_trama_ler == 0) {
                        trama_inicial(RR_PERDIDO, FR_A_TX, FR_C_RR0);
                }

                tcflush(port, TCIOFLUSH);
```

```
                if (write(port, RR_PERDIDO, 5) < 0) {
                    printf("Trama duplicada devido a RR perdido");
                    return -3;
                }
                memset(trama_recebida, 0, trama_length );
                return -2;

            default:
                break;
        }
    }

    return (aux_trama - 6);

}
//LLWRITE PRONTO
int llwrite(int port, unsigned char* buffer, int length) {

    int estado = 0, aux = 0, time_out_cont = 0, ciclo1 = 0, enviado = 0, trama_I
= 0;
    unsigned char RR[5], REJ[5];
    unsigned char trama_enviada[trama_length];
    unsigned char trama_recebida[trama_length];

    //PARA CADA TRAMA RR E REJ TEMOS 2 CASOS EM QUE O N=1 E N=0
    if(N_trama_escrever == 1) {
        trama_inicial(RR, FR_A_TX,FR_C_RR0);
        trama_inicial(REJ, FR_A_TX, FR_C_REJ1);
    } else if(N_trama_escrever == 0) {
        trama_inicial(RR, FR_A_TX, FR_C_RR1);
        trama_inicial(REJ, FR_A_TX, FR_C_REJ0);
    } else {
        return -1;
    }

    trama_I  = tramatipo_I(trama_enviada, buffer, length, N_trama_escrever); //cr
iar uma trama do tipo I

    while(!ciclo1) {
        switch(estado) {
            case 0:
                enviado = write(port, trama_enviada, trama_I);
                estado = 1;
                break;

            case 1:
```

```
            //vê se recebe alguma coisa na porta
                aux = ler_trama(port, trama_recebida);
                if(aux==-5) return -5; // cabo com má conecção
                if(aux == erro) {
                    if(time_out_cont < 3) {
                        printf("Não recebemos nada na porta");
                        sleep(1);
                        time_out_cont++;
                        estado = 0;
                    } else {
                        printf("time out excedido - 3 tentativas falhadas");
                        return -1;
                    }
                } else {
                    trama_cont++;
                    estado = 2;
                }
                break;

            case 2:
                //rr RECEBIDO
                if(memcmp(RR, trama_recebida, 5) == 0) {
                    N_trama_escrever  = 1 - N_trama_escrever ;
                    ciclo1 = 1;
                } else if(memcmp(REJ, trama_recebida, 5) == 0){   //COMPARA SE É
UM rej E É ENVIADA NOVAMENTE A TRAMA I
                    printf("\n\nREJ RECEBIDO. TRAMA I ENVIADA NOVAMENTE\n\n"); //
QUANDO HÁ ERRO - CABO DESLIGADO
                    estado = 0;
                } else {
                    printf("\n\n nada recebido\n\n");
                    time_out_cont++;
                    estado = 0;
                }
                break;

            default:
                break;
        }
    }

    return enviado;

}
//LLCLOSE PRONTO
int llclose(int port, int Tx_Rx) { // Fecha a logação entre os 2 pontos
```

```
    int estado = 0, aux = 0, time_out_cont = 0, ciclo1 = 0, aux_trama = 0;
    unsigned char DISC[5], UA[5];
    unsigned char trama_recebida[trama_length];

    trama_inicial(DISC, FR_A_TX, FR_C_DISC);
    trama_inicial(UA, FR_A_TX, FR_C_UA);

    if (Tx_Rx == TX) {
        while (!ciclo1) {
            switch (estado) {
                case 0:
                //Envia o DISC
                    tcflush(port, TCIOFLUSH);
                    aux = write(port, DISC, 5);
                    if( aux< 0) {
                        printf("\nErro na escrita do DISC para a porta\n");
                        return -1;
                    }
                    estado = 1;
                    break;

                case 1:
                //RECEBER DISC
                    aux_trama = ler_trama(port, trama_recebida);
                    if(aux_trama==-5) return -5; // cabo com má conecção
                    if(aux_trama == erro) {
                        if(time_out_cont < 3) {
                            printf("\ndisc não foi recebido no RX\n");
                            sleep(1);
                            time_out_cont++;
                            estado = 0;
                        } else {
                            printf("\ndisc não foi recebido no TX após as 3 tenta
tivas.\n");

                            return -1;
                        }
                    } else {

                        if(memcmp(trama_recebida, DISC, 5) == 0) {
                            estado = 2;
                        } else {
                            estado = 0;
                        }
                    }
                    break;
```

```
                case 2:
                //ENVIAR UA Sabendo que foi recebido o DISC
                    tcflush(port, TCIOFLUSH);
                    aux = write(port, UA, 5);
                    if(aux < 0) {
                        printf("\nUA NAO FOI ESCRITO NA PORTA\n");
                        return -1;
                    }
                    printf("\nConexão terminada. TOP!\n");
                    ciclo1 = 1;
                    break;

                default:
                    break;
            }
        }
        return 0;
    } else if (Tx_Rx == RX) {
        while (!ciclo1) {
            switch (estado) {
                case 0:
                //Envia o DISC
                    tcflush(port, TCIOFLUSH);
                    aux = write(port, DISC, 5);
                    if( aux< 0) {
                        printf("\nErro na escrita do DISC para a porta\n");
                        return -1;
                    }
                    estado = 1;
                    break;

                case 1:
                //RECEBER UA
                    aux_trama = ler_trama(port, trama_recebida);
                    if(aux_trama==-5) return -5; // cabo com má conecção
                    if(aux_trama == erro) {
                        if(time_out_cont < 3) {
                            printf("\nUA não foi recebido no RX.\n");
                            sleep(1);
                            time_out_cont++;
                            estado = 0;
                        } else {
                            printf("\nUA não foi recebido no RX após esaux_tramaa
das as 3 tentativas.\n");
                            return -1; //ultrapassadas as 3 tentativas
```

```
                        }
                    } else {
                        if(memcmp(trama_recebida, UA, 5) == 0) { //comparar os pr
imeiros 5 bits da trama e do UA (se =0 é igual)
                            printf("\nConexão terminada. TOP\n");
                            ciclo1 = 1;
                        } else {
                            estado = 0;
                        }
                    }
                    break;

                default:
                    break;
            }
        }

    return 0;
    }

    return -1;

}
```

3. Camada_de_ligação.h

```
int llopen(int port, int Tx_Rx); // Cria a ligação entre os 2 pontos
int llwrite(int port, unsigned char* buffer, int length); // Vai ser retornado o
comprimento da trama sem os comprimentos dos cabeçalhos
int llread(int port, unsigned char* buffer);
int llclose(int port, int Tx_Rx); // Fecha a logação entre os 2 pontos
```

4. Biblio.c

```
#include <sys/types.h>
```

```c
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <signal.h>
#include <inttypes.h>
#include "biblio.h"

int ler_trama(int port, unsigned char *trama) {// le a trama da porta

    int done = 0, res = 0, b = 0, cont=0;
    unsigned char get;
    int aux=0;
    while(!done) {
        read_verificacao:   aux=read(port, &get, 1); // verifica quantos bytes fo
ram lidos
        if(aux<0) return -5; // deu erro para
        if(aux==0){ // leu zero
            cont++;
            sleep(0.01);
            if(cont==30) return -1;
            goto read_verificacao;
        }
        //*** Nesta zona da funcao retorna um erro ou detem 1 byte ***//
        if(get == FR_F) {
            if(b == 0) {
                trama[b++] = get;
            } else {
                if(trama[b-1] == FR_F) {
                    memset(trama, 0, trama_length );
                    b = 0;
                    trama[b++] = FR_F;
                } else {
                    trama[b++] = get;
                    done = 1;
                }
            }
        } else {
            if(b > 0) {
                trama[b++] = get;
            }
```

```
        }
    }

    return b;  //retorna o tamanho da trama

}

void trama_inicial(unsigned char *trama, unsigned char A, unsigned char C) {// fo
rma a trama inicial -- camada data_link || introduz na trama

    trama[0] = FR_F;
    trama[1] = A;
    trama[2] = C;
    trama[3] = trama[1]^trama[2];
    trama[4] = FR_F;

}

int tramatipo_I(unsigned char* trama, unsigned char* buff, int tamanho, int N_tra
ma_escrever) { // onde é feito o byte stuffing

    int b=0, i=0;
    //TRAMAS DE CONTROLO SET E UA DO TIPO [F A C Bcc F]
    unsigned char BCC2 = 0;
    //Flag inicial
    trama[b++] = FR_F;
    //A
    trama[b++] = FR_A_TX;

    if(N_trama_escrever == 0) {                 //C
        trama[b++] = FR_C_SEND0;
    } else if(N_trama_escrever == 1) {
        trama[b++] = FR_C_SEND1;
    }

    trama[b++] = trama[1]^trama[2];    // BCC1=A^C  calculo do BCC1

            //Byte stuffing BCC1
    for(i = 0; i < tamanho; i++){
        BCC2 = BCC2 ^ buff[i];              //BCC2=D1^D2^D3^... ^Dn  calculo do B
CC2
        if(buff[i] == FR_F) {
            trama[b++] = ESC;
            trama[b++] = FR_F_AUX;
        } else if(buff[i] == ESC) {
            trama[b++] = ESC;
```

```
            trama[b++] = ESC_AUX;
        } else {
            trama[b++] = buff[i];
        }
    }
        //Byte stuffing BCC2
    if(BCC2 == FR_F) {
        trama[b++] = ESC;
        trama[b++] = FR_F_AUX;
    } else if(BCC2 == ESC) {
        trama[b++] = ESC;
        trama[b++] = ESC_AUX;
    } else {
        trama[b++] = BCC2;
    }

    trama[b] = FR_F;

    return b+1; // retorna o tamanho da trama
}

int construir_pacoteTVL(unsigned char C, unsigned char* pacote, parametros* tvl)
{ //preenche pacote com o campo C, os TLVs de tvl e retorna o seu tamanho como in
teiro

    int i = 0, j = 0, b = 0;

    pacote[b++] = C;
    for(i = 0; i < tamanho_trama; i++) {
        pacote[b++] = tvl[i].T;
        pacote[b++] = tvl[i].L;
        for(j = 0; j < (int) tvl[i].L; j++) {
            pacote[b++] = tvl[i].V[j];
        }
    }

    return b;  // retorna o seu tamanho como inteiro

}

int construir_pacoteDados(unsigned char* buff, unsigned char* pacote, int tamanho
_pacote, int* seq_num) { //coloca os dados de buff, com tamanho tamanho_pacote, n
um pacote pacote com um número de sequência seq_num passado por referência, e inc
rementa-o
    //  [0 1 2 3 4]=[C N L2 L1 P(DADOS)]
    int i = 0, x = 0;
```

```
    pacote[0] = C_DADOS;
    pacote[1] = (char) (*seq_num)++;
    if((*seq_num) == 256) {
        *seq_num = 0;
    }

    x = tamanho_pacote % 256;
    pacote[2] = (tamanho_pacote - x) / 256;
    pacote[3] = x;

    for(i = 0; i < tamanho_pacote; i++) {
        pacote[i+4] = buff[i];
    }
    return i+4;

}

void pacote_tvl(unsigned char *pacote, parametros *tvl) {  //preenche os TLVs de
tvl com os dados em pacote

    int i = 0, j = 0, tamanho = 0, k = 0;
    while (k < tamanho_trama) {
        tvl[k].T = pacote[i];
        tvl[k].L = pacote[++i];
        tamanho = (int)(tvl[k].L);
        tvl[k].V = (unsigned char *) malloc(tamanho);
        for (j = 0; j < tamanho; j++){
            tvl[k].V[j] = pacote[++i];
        }
        i++;
        k++;
    }

}

void pacotedados(unsigned char *pacote, data *pacote_data) { //alterar o pacote p
ara uma estrutura pacote_data, do tipo data

    int i=0;
    int j=0;

    (*pacote_data).N = pacote[0]; //N
    (*pacote_data).L2 = pacote[1]; //L2
    (*pacote_data).L1 = pacote[2]; //L1
    (*pacote_data).dados = (unsigned char*) malloc(256*(int)pacote[1] + (int)paco
te[2]);
```

```
    for (j = 0, i = 3; j < 256*(int)(*pacote_data).L2 + (int)(*pacote_data).L1; j
++, i++) { //DADOS
        (*pacote_data).dados[j] = pacote[i];
    }

}

int porta(char *port, struct termios *oldtio) { // set da porta e retorna o seu a
pontador

    if((strcmp("/dev/ttys002", port) != 0) && (strcmp("/dev/ttys003", port) != 0)
) {
        perror("changePort(): wrong argument for port");
        return -1;
    }

    /*
        Open serial port device for reading and writing and not as controlling tt
y
        because we don't want to get killed if linenoise sends CTRL-C.
    */
    struct termios newtio;

    int fd;
    if ((fd = open(port, O_RDWR | O_NOCTTY )) < 0) {
        perror(port);
        return -1;
    }

    if ( tcgetattr(fd, oldtio) == -1) { /* save current port settings */
        perror("tcgetattr");
        return -1;
    }

    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = 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]    = 1;   /* inter-character timer unused (estava a 0)*/
```

```
    newtio.c_cc[VMIN]     = 0;   /* blocking read until 5 chars received (estava
a 5)*/

    /*
        VTIME e VMIN devem ser alterados de forma a proteger com um temporizador
a
        leitura do(s) proximo(s) caracter(es)
    */

    tcflush(fd, TCIOFLUSH);

    if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
        perror("tcsetattr");
        return -1;
    }

    printf("\nNew termios structure set\n");

    return fd; //retorna o seu apontador

}

int getFileLength(int fd) { // retorna o tamanho do ficheiro

    int length = 0;

    if((length = lseek(fd, 0, SEEK_END)) < 0) {
        perror("lseek()");
        return -1;
    }
    if(lseek(fd, 0, SEEK_SET) < 0) {
        perror("lseek()");
        return -1;
    }

    return length;

}
```

5. Biblio.h

```
#include <inttypes.h>
#include <sys/types.h>
#include <sys/stat.h>


#define BAUDRATE B115200
#define TX 1
#define RX 0

#define TAM_BUF       65539           // k = 256 × (28 − 1) + (28 − 1) = 65535 +
4 (C, N, L2 e L1)
//tamanho_max  de uma trama: (65539 + 1) × 2 + 5 = 131085 devido a efeitos de byt
e stuffing e cabeçalho da camada de ligação de dados
#define trama_length      TAM_BUF*2+7     //tam max trama
#define tamanho_trama            2 //nome e tamanho

#define FR_F          0x7E
#define FR_F_AUX      0x5E
#define ESC           0x7D
#define ESC_AUX       0x5D
#define FR_A_TX       0x03    //Comandos enviados pelo Emissor e Respostas enviad
as pelo Receptor
#define FR_A_RX       0x01    //Comandos enviados pelo Receptor e Respostas envia
das pelo Emissor
#define FR_C_SET      0x03
#define FR_C_DISC     0x0B
#define FR_C_UA       0x07
#define FR_C_RR0      0x05
#define FR_C_RR1      0x85
#define FR_C_REJ0     0x01
#define FR_C_REJ1     0x81
#define FR_C_SEND0    0x00    //Campo de Controlo que recebe da trama I e depois
altera 0
#define FR_C_SEND1    0x40    //Campo de Controlo que recebe da trama I e depois
altera 1
#define C_START       0x02
#define C_DADOS       0x01
#define C_END         0x03

#define erro  -1


typedef struct {
```

```
    unsigned char* nome;
    int tamanho;
} detalhes;

//SLIDE 23
typedef struct {
    unsigned char T; //Type
    unsigned char L; //Length
    unsigned char *V; //Value
} parametros;


                    //N – número de sequência (módulo 255)
                    //L2 L1 –indica o número de octetos(K) do campo de dados
                    //P1 ... PK – campo de dados do pacote (K octetos)

typedef struct { //pacote de dados
    unsigned char N;
    unsigned char L1;
    unsigned char L2;
    unsigned char *dados;
} data;


int ler_trama(int port, unsigned char *trama);// le a trama
void trama_inicial(unsigned char* trama, unsigned char A, unsigned char C);// for
ma a trama inicial -- camada data_link || introduz na trama
int tramatipo_I(unsigned char* trama, unsigned char* buff, int length, int sendNu
mber);// onde é feito o byte stuffing
int construir_pacoteTVL(unsigned char C, unsigned char* pacote, parametros* tvl);
 //preenche pacote com o campo C, os TLVs de tvl e retorna o seu tamanho como int
eiro
int construir_pacoteDados(unsigned char* buff, unsigned char* pacote, int tamanho
_pacote, int* seq_num);//coloca os dados de buff, com tamanho tamanho_pacote, num
 pacote pacote com um número de sequência seq_num passado por referência, e incre
menta-o
void pacote_tvl(unsigned char *pacote, parametros *tvl); //preenche os TLVs de tv
l com os dados em pacote
void pacotedados(unsigned char *pacote, data *pacote_data); //alterar o pacote pa
ra uma estrutura pacote_data, do tipo data
int porta(char *port, struct termios *oldtio); // set da porta e retorna o seu ap
ontador
int getFileLength(int fd);// retorna o tamanho do ficheiro
```