U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

1º Trabalho Laboratorial

Ligação de Dados

Grupo 7

Trabalho realizado por:

Hugo Gomes – up202004343

João Moreira – up202005035

Unidade Curricular:

RC

Ano Letivo:

2022/2023

Sumário

No âmbito da unidade curricular de Redes de Computadores, foi nos proposto a implementação de um protocolo de ligação de dados, para a transferência de ficheiros de um computador para outro, através de uma porta série.

Assim, podemos afirmar que concluímos todos os objetivos propostos, obtendo uma aplicação robusta capaz de transferir ficheiros sem perda de dados.

Introdução

Este trabalho tem como principal objetivo a implementação de um protocolo de ligação de dados de acordo com as informações descritas no guião.

Neste guião, foi nos pedido que a nossa implementação seguisse algumas características especificas, entre as quais: independência entre camadas, isto é, separação total entre a camada de ligação de dados e a camada de aplicação, tendo, esta última, total desconhecimento dos detalhes da camada de ligação, usando-a apenas como um serviço, confirmação e controlo de erros através de uma variante de Stop and Wait, utilização da técnica de byte stuffing entre outros.

Quanto ao relatório, este tem como objetivo a exposição da componente teórica deste projeto, tendo a seguinte estrutura:

- **Arquitetura-** blocos funcionais e interfaces.
- **Estrutura do código** – APIs, principais estruturas de dados, principais funções e a sua relação com a arquitetura.
- **Casos de uso principais** – identificação e sequências de chamada de funções.
- **Protocolo de ligação lógica** - identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes aspetos.
- **Protocolo de aplicação** - identificação dos principais aspetos funcionais, descrição da estratégia de implementação destes aspetos.
- **Validação** - descrição dos testes efetuados.
- **Eficiência do protocolo de ligação de dados** - caraterização estatística da eficiência do protocolo, efetuada recorrendo a medidas sobre o código desenvolvido.
- **Conclusões** - Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

Arquitetura

O projeto está organizado em dois blocos funcionais: um associado ao computador que envia o ficheiro (sender) e outro ao computador que recebe o ficheiro (receiver). Os ficheiros receiver.c e receiver.h contêm o código do receiver enquanto os ficheiros sender.c e sender.h o código correspondente ao sender. Em cada um destes blocos funcionais, existe uma separação clara entre a camada de aplicação e a camada de ligação de dados.

A camada de ligação de dados tem como objetivo fornecer um serviço de comunicação fiável, entre dois sistemas por meio de um canal. Nesta camada, encontramos funções genéricas do protocolo de ligação de dados que fornecem as funcionalidades de sincronismo de tramas (framing), numeração de tramas, controlo de erros, controlo de fluxo e estabelecimento/término da ligação.

A camada de aplicação utiliza as funções da camada de ligação de dados como um serviço para transferência da informação. Esta é responsavel por enviar o ficheiro, dividindo-o em pacotes de informação que são passados à camada de ligação de dados.

Estrutura do código

O código encontra-se dividido em dois ficheiros: o sender.c que corresponde ao writer e o receiver.c que corresponde ao reader. Ambos estes ficheiros chamam funções da camada da aplicação que, por sua vez, chama funções da camada da ligação de dados.

Receiver:

Funções da camada de ligação:

- **LLOPEN()** - Recebe trama SET e envia trama UA;
- **LLREAD()** - Lê trama de informação e realiza destuffing;
- **LLCLOSE()** - Recebe trama DISC e envia trama DISC;

Funções camada de aplicação

- **createFile() -** Cria o ficheiro através da informação recebida;
- **isEndPacket() -** Verifica se o packet recebido é o END packet;
- **removeControlHeader() -** Remove os bytes associados ao controlo do packet obtendo apenas os dados;
- **main() -** Função principal da camada de aplicação que utiliza todas as outras funções para abrir a ligação, ler um packet de cada vez até receber o END packet, construir o ficheiro com os packets recebidos e fechar a ligação;

Variáveis globais

- **volatile int STOP;**
- **int expectedFrame;**
- **struct termios oldtio;**

- **struct termios newtio;**

Macros pertinentes:

- **BAUDRATE**

Sender:

Funções da camada de ligação de dados:

- **LLOPEN()** - Envia trama SET e recebe trama UA;
- **LLWRITE()** - Constrói uma trama de informação com os dados recebidos como argumento, realiza byte stuffing e tenta enviar esta mesma trama;
- **LLCLOSE()** - Envia trama DISC, recebe trama DISC e envia trama UA;

Funções da camada de aplicação:

- **openFile()** - Abre o ficheiro, calcula o tamanho do ficheiro e lê todos os bytes do mesmo para um buffer;
- **createControlPacket()** - Cria START packet, caso start = True. Caso contrário, criar END packet;
- **createPacket()** - Cria packet de informação, através do buffer de bytes do ficheiro;
- **addHeaderPacket()** - Adiciona header com campo de controlo, número de sequência, número de octetos a enviar ao packet de dados;
- **main()** - Função principal da camada de aplicação que utiliza todas as outras funções para abrir a ligação, abrir o ficheiro, construir os packets de controlo/informação, enviá-los através da camada de ligação de dados e fechar a ligação;

Variáveis globais

- **int alarmEnabled;**
- **int alarmCount;**
- **int UAreceived;**
- **int currentFrame;**
- **struct termios oldtio;**
- **struct termios newtio;**

Macros pertinentes

- **BAUDRATE**
- **TIMEOUT**
- **MAX_SENDS**

Casos de Uso Principais

A nossa aplicação permite ao utilizador transmitir um ficheiro de um computador para outro, através de uma porta série. Em primeiro lugar, o utilizador precisa de compilar a aplicação a partir de um Makefile. De seguida, o computador que pretende enviar o ficheiro (sender) pode executar o programa da seguinte forma: ./sender [porta-série] [nome ficheiro] Ex: ./sender /dev/ttyS10 pinguin.gif. Por outro lado, o computador que pretende receber o ficheiro (receiver) pode executar o programa da seguinte forma: ./receiver [porta-sérire] Ex:/dev/ttyS10.

O computador receiver deverá ser o primeiro a ser executado para esperar que o transmiter envie as tramas de estabelecimento de ligação. Caso contrário, o transmiter iniciará as tentativas de envio das tramas e terminará quando o número de tentativas for excedido.

Após a ligação ser estabelecida, por cada packet enviado, uma mensagem de envio ou receção surge na consola. Caso ocorram erros, as seguintes mensagens surgem na consola:

- Erro de BCC1 -> Error in the protocol;
- Erro de BCC2 -> Error in the data;
- Frame duplicado -> Duplicate Frame;
- Reject recebido -> REJ received;

Protocolo de ligação lógica

Máquinas de Estado:

Antes de abordamos a nossa implementação das funções principais é necessário explicar como implementamos as máquinas de estado. ///TALVEZ METER FOTO DAS MAQUINAS DE ESTADOS:

No recetor, criamos uma máquina de estados geral que é capaz de receber uma trama UA ou uma trama DISC. A função receiveControlWord recebe o file descriptor da porta série e um unsigned char que representa quais das tramas de supervisão estamos a tentar ler UA ou DISC.

No emissor, criamos uma máquina de estados geral que é capaz de receber uma trama UA, REJ, RR ou DISC. A função receiveControlWord recebe o file descriptor da porta série e um apontador para um unsigned char onde será devolvido o tipo de trama recebido.

LLOPEN

```
int LLOPEN (int fd);
```

Esta função é responsável pela fase de estabelecimento da ligação. No recetor, utilizando a função receiveControlWord tentamos ler a trama SET. De seguida, caso a receção seja confirmada, enviamos a trama UA.

No emissor, a trama SET é enviada e um temporizador é ativado. Caso o temporizador dispare, um reenvio da trama SET é executado. As tentativas de reenvio terminam após um número pré-definido de tentativas ser excedido ou a trama UA ser recebida. Caso este ciclo termine pelo número máximo de tentativas, a função retorna FALSE, caso contrário retorna TRUE.

LLREAD

int LLREAD(int fd, unsigned char * messageReceived);

Esta função, apenas, está presente no recetor e é responsável por fazer a leitura de uma trama e o destuffing da mesma. De modo a fazer a leitura caracter a caracter, criamos uma maquina de estados. //FOTO MAQUINA DE ESTADOS??

Esta máquina de estados inicia no estado 0. Neste estado, verificamos a receção da FLAG. Caso se verifique, passamos para o estado um onde confirmamos a receção do A e, consequentemente, saltamos para o estado dois. No estado dois, verificamos o campo de controlo C e guardamos este para, mais tarde, no estado três, fazermos a verificação do BCC1. Caso o BCC1 esteja correto, passamos ao estado 4 onde vamos começar a receber os dados. Se o caracter recebido for uma flag, significa que a trama chegou ao fim e saltamos para o estado 6, caso o caracter seja um ESC teremos que realizar o destuffing, executado no estado 5. Caso contrário, adicionamos o caracter recebido ao buffer onde guardamos a mensagem recebida. Por fim, no estado 6, é calculado o BCC2 para os dados que recebemos e comparado com o caracter recebido antes da FLAG. Caso este se confirme, enviamos um received ready, caso contrário, enviamos um reject.

Esta função retorna o tamanho da mensagem recebido, caso todas as condições estejam corretas. Caso alguma falhe, retorna -1. O apontador para o buffer de dados é retornado no parâmetro messageReceived.

LLWRITE

int LLWRITE (int fd, unsigned char* msg, int size);

Esta função apenas está presente no emissor e é responsável por construir uma trama com a mensagem recebida, fazer o stuffing da mesma e enviá-la.

Primeiro, é criado um buffer com o tamanho da mensagem mais 6 bytes. Estes 6 bytes são utilizados para os campos de controlo do protocolo de ligação de dados (FLAG,A,C,BCC1). De seguida, realizamos stuffing nos dados, calculamos o BCC2 dos dados sem stuffing e fazemos stuffing do BCC2. Por fim, adicionamos a FLAG final e finalizamos a criação da trama.

Deste modo, o passo seguinte é enviar a trama. O envio da trama de informação é semelhante ao envio da trama de supervisão, isto é, tentamos enviar a trama, iniciamos um temporizador e iniciamos a state machine de leitura. Caso a palavra de controlo recebida seja um Received Ready, a função termina e retorna o tamanho da mensagem enviada. Caso seja um Reject, se este foi provocado pelo envio de um frame duplicado, a função termina alterando o número da trama a enviar. Caso contrário, o temporizador é reiniciado, o número de tentativas de envio volta a zero e uma nova tentativa de envio é efetuada. As tentativas de envio terminam após um número de tentativas máximo ser excedido ou um Received Ready ser recebido.

LLCLOSE

int LLCLOSE (int fd);

Esta função é responsável pela fase de terminação da ligação. No recetor, utilizando a função receiveControlWord, tentamos ler a trama DISC. De seguida, caso a receção seja confirmada, enviamos a trama DISC.

No emissor, a trama DISC é enviada e um temporizador é ativado. Caso o temporizador dispare, um reenvio da trama DISC é executado. As tentativas de reenvio terminam após um número de tentativas ser excedido ou a trama DISC ser recebida. Após a receção do disc ou o número de tentativas de reenvio ser atingido, um UA é enviado.

Em ambos os programas, é reposta a configuração inicial da porta série, restaurando os valores iniciais alterados no LLOPEN.

Protocolo de aplicação

Em ambas as máquinas, no início da função main, é feita uma verificação dos argumentos recebidos pelo utilizador. Caso haja algum erro, uma mensagem com um exemplo de utilização é apresentada.

No emissor, o programa começa por abrir o ficheiro para envio, calcula o tamanho do mesmo e lê todos os seus bytes. A execução continua com a chamada da função LLOPEN para estabelecer a ligação. De seguida, é necessário criar o START PACKET. Esta ação é realizada pela função createControlPacket com o parâmetro start a TRUE. O retorno desta função é passado à função LLWRITE. Enquanto o ficheiro não terminar, a função createPacket é chamada para criar um packet com uma parte do mesmo. Através da função addHeaderPacket, o header da camada de aplicação é concatenado ao packet de dados anterior e este é passado à função LLWRITE para envio. Por fim, novamente utilizando a função createControlPacket, mas com o paramentro start a FALSE, o END PACKET é criado e enviado com o LLWRITE.

No recetor, primeiramente, a função LLOPEN é chamada para o estabelecimento da ligação. De seguida, utilizamos a função LLREAD para receber o START PACKET e retirar o tamanho do ficheiro do mesmo. Posteriormente, enquanto o END PACKET não for recebido, lemos um packet, retiramos o header deste através da função removeControlHeader e adicionamo-lo ao buffer com a informação recebida. Por fim, chamamos a função createFile que cria um ficheiro com toda a informação recebida.

No fim dos dois programas, a função LLCLOSE é evocada para iniciar a fase de término da ligação.

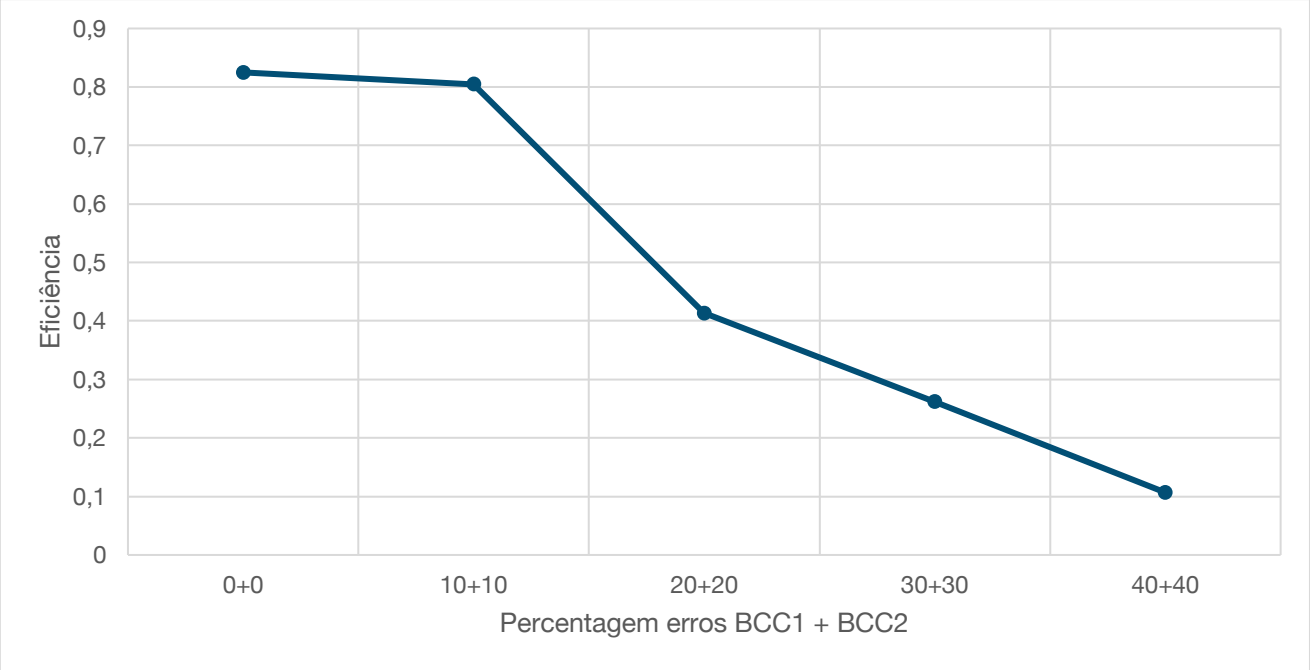
Validação

De modo a validar a robustez do nosso programa, efetuamos um conjunto de testes. Começamos por interromper a ligação da porta série, fazendo com que ocorra um timeout no emissor e uma tentativa de reenvio. De seguida, testamos a forma como o programa lida com cada tipo de erro, quer gerando curto circuito na porta série, quer introduzindo erros simulados, através das funções errorBCC1 e errorBCC2. Por fim, alteramos o tamanho dos pacotes e o tamanho do ficheiro. Todos estes testes foram realizados com sucesso.

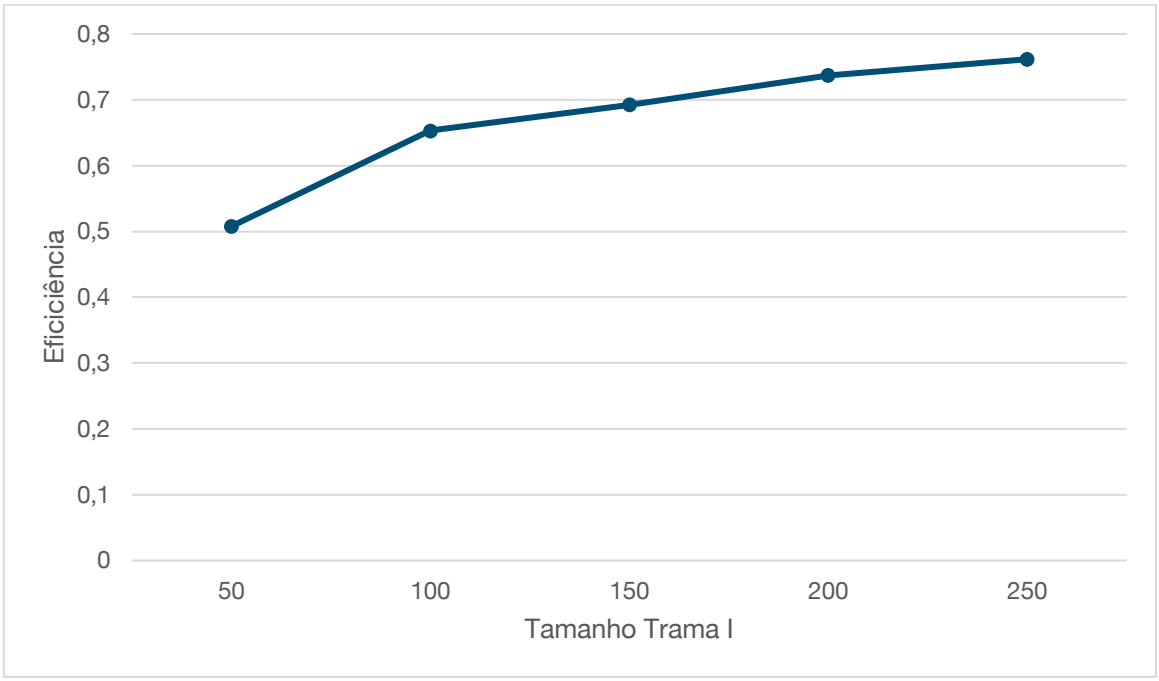
Eficiência do protocolo de ligação de dados

O cálculo da eficiência é obtido dividindo o débito(R) pela capacidade de transmissão. Para calcular o débito R, medimos o tempo de execução do nosso programa e dividimos pelo tamanho do ficheiro a enviar. Deste modo, elaboramos 3 gráficos variando vários fatores que influenciam a eficiência.

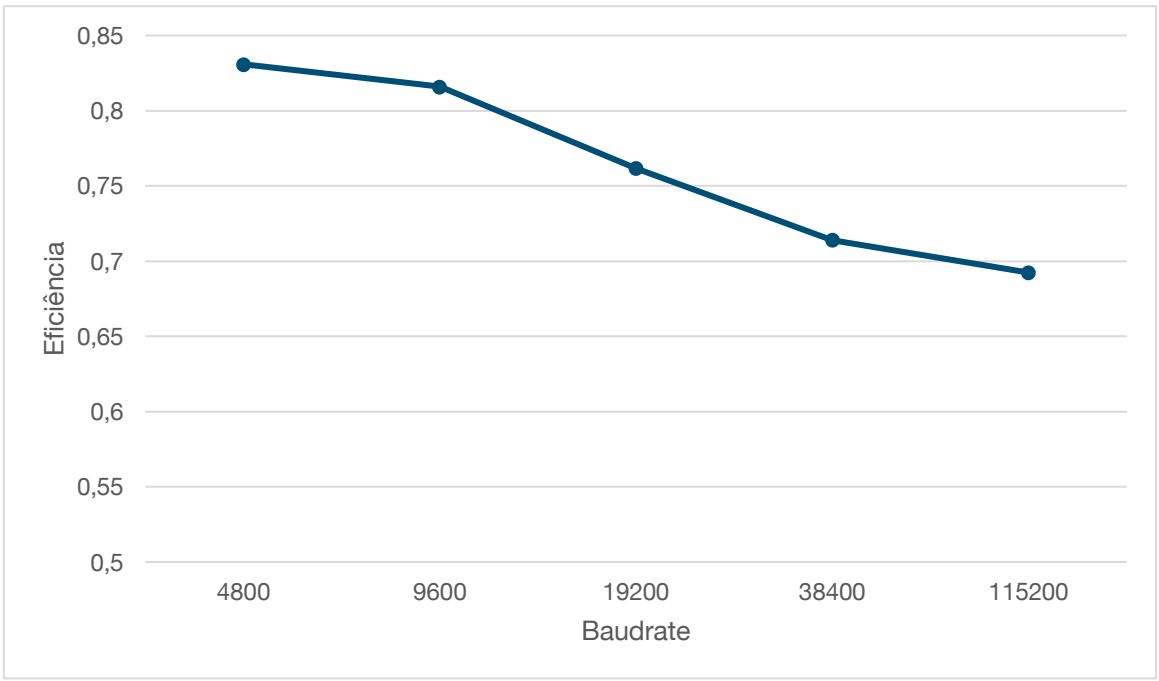
No primeiro, variamos a percentagem de erros gerados e concluímos que, com o aumento da percentagem de erros, a eficiência diminui.



No segundo, variamos o tamanho das tramas de informação e concluímos que, com o aumento do tamanho das tramas, a eficiência aumenta.



No terceiro, variamos a capacidade de ligação (baudrate) e concluímos que a eficiência diminui ligeiramente com o aumento do baudrate.



Conclusões

O tema deste trabalho é o protocolo de ligação de dados, que consiste em fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão, neste caso em concreto, um cabo série.

Adicionalmente, foi dado a conhecer o termo independência entre camadas, e cada um dos blocos funcionais da arquitetura da aplicação desenvolvida, writer e reader, cumpre esta independência. Na camada de ligação de dados, não é feita qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de Informação. No que diz respeito à camada de aplicação, esta não conhece os detalhes do protocolo de ligação de dados, apenas a forma como o serviço é acedido.

Em suma, o trabalho foi concluído com sucesso, tendo-se cumprido todos os objetivos. A sua elaboração contribuiu positivamente para um aprofundamento do conhecimento, tanto teórico como prático, do tema em questão.

Anexos

Sender.h

```
#ifndef SENDER_HEADER
#define SENDER_HEADER

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termios.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>

#define BAUDRATE B38400
#define _POSIX_SOURCE 1 // POSIX compliant source

#define FALSE 0
#define TRUE 1

#define FLAG (0x7E)
#define A (0x03)
#define A_1 (0x01)
#define SET (0x03)
#define UA (0x07)
#define DISC (0x0B)
#define C0 (0x00)
#define C1 (0x40)
#define ESC (0x7D)
#define FLAGESC (0x5E)
#define FLAGFLAGESC (0x5D)
#define RR0 (0x05)
#define RR1 (0x85)
#define REJ0 (0x01)
#define REJ1 (0x81)

//packet Application layer
#define CP_START (0x02)
#define CP_END (0x03)
#define CP_TYPE_FILESIZE (0x00)
#define CP_LENGTH_FILESIZE (0x04)
#define Control_DATA_PACKET (0x01)
#define BUF_SIZE 256
#define PACKET_SIZE 500

#define TIMEOUT 3 //Tempo até Timeout
#define MAX_SENDS 3 //Numero maximo de tentativas de envio

#define BCC1ERROR 0
#define BCC2ERROR 0


//--------------------Data link layer--------------------------------

//Função que cria uma trama de supervisão e envia
void sendControlWord(int fd, unsigned char C);

//Função de alarme
void alarmHandler(int signal);
```

```
//--------------------Data link layer--------------------------------

//Função que cria uma trama de supervisão e envia
void sendControlWord(int fd, unsigned char C);

//Função de alarme
void alarmHandler(int signal);

//Função com máquina de estados para leitura de controlWord
void receiveControlWord(int fd, unsigned char * cReceived);

//Função de envio das tramas
int LLWRITE(int fd, unsigned char* msg, int size);

//Função de terminação da ligação
int LLCLOSE(int fd);

//Função de estabelecimento da ligação
int LLOPEN(int fd);


//--------------------Application Layer---------------------------

//Função que abre um ficheiro, lê o seu conteudo e retira o seu tamanho
unsigned char* openFile(unsigned char *fileName, int* sizeFile);

//Função que cria um packet de Controlo
unsigned char *createControlPacket(int start,int fileSize,int *sizeControlPacket);

//Função que adiciona o cabeçalho da application layer
unsigned char* addHeaderPacket(unsigned char* packet, int fileSize, int packetSize);

//Função que cria um packet a partir dos dados do ficheiro
void createPacket(unsigned char* fileData,int*indexFile, int fileSize,int* packetSize, unsigned char* packet);


//Função que introduz erros no BCC1
unsigned char *errorBCC1(unsigned char* packet, int packetSize);

//Função que introduz erros no BCC2
unsigned char *errorBCC2(unsigned char* packet, int packetSize);

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

#endif //SENDER_HEADER
```

Sender.c

```c
#include "sender.h"

struct termios newtio;
struct termios oldtio;

//alarm variables
int alarmEnabled = FALSE;
int alarmCount = 0;

int UAreceived = FALSE;
int currentFrame = 0;

int sequenceNumberPacket = 0;

void sendControlWord(int fd, unsigned char C){
    unsigned char message[5];
    message[0] = FLAG;
    if(C == DISC || C == UA){
        message[1] = A_1;
    }else{
        message[1] = A;
    }
    message[2] = C;
    message[3] = message[1] ^ message[2];
    message[4] = FLAG;
    write(fd,message,5);
}

void alarmHandler(int signal)
{
    alarmEnabled = FALSE;
    alarmCount++;

    printf("time-out #%i\n", alarmCount);
}
```

```c
//state Machine receive UA REJ0 REJ1 RR0 RR1 DISC
void receiveControlWord(int fd, unsigned char * cReceived){
    int state = 0;
    unsigned char c;
    unsigned char A_check;
    while(state != 5 && alarmEnabled){
        read(fd, &c,1);
        switch(state){
            case 0:
                if(c == FLAG){
                    state = 1;
                }
                break;
            case 1:
                if(c == A || c == A_1){
                    state = 2;
                }else{
                    if (c == FLAG){
                        state = 1;
                    }else{
                        state = 0;
                    }
                }
                break;
            case 2:
                if(c == UA || c == REJ0 || c == REJ1 || c == RR0 || c == RR1 || c == DISC){
                    state = 3;
                    *cReceived = c;
                }else{
                    if(c == FLAG)
                        state = 1;
                    else
                        state = 0;
                }
                break;
            case 3:

                if(*cReceived == DISC){
                    A_check = A_1;
                }else{
                    A_check = A;
                }

                if(c == (A_check ^ *cReceived))
                    state = 4;
                else{
                    *cReceived = 0xFF; //arbitary value
                    state = 0;
                }
                break;
            case 4:
                if(c == FLAG){
                    state = 5;
                    alarm(0);
                }
                else
                    state = 0;
                break;
        }
    }
}
```

```c
int LLWRITE(int fd, unsigned char* msg, int size){
    unsigned char *frameFinal = (unsigned char *)malloc((size + 6) * sizeof(unsigned char));
    int sizeFrameFinal = size + 6;

    int rejected = FALSE;

    frameFinal[0] = FLAG;
    frameFinal[1] = A;

    if(currentFrame == 0){
        frameFinal[2] = C0;
    }else{
        frameFinal[2] = C1;
    }

    frameFinal[3] = frameFinal[1] ^ frameFinal[2];

    int k = 4;

    //Stuffing data
    for(int i = 0; i < size; i++){
        if(msg[i] == FLAG){
            sizeFrameFinal++;
            frameFinal = (unsigned char *)realloc(frameFinal,sizeFrameFinal);
            frameFinal[k] = ESC;
            frameFinal[k + 1] = FLAGESC;
            k += 2;
        }else if(msg[i] == ESC){
            sizeFrameFinal++;
            frameFinal = (unsigned char *)realloc(frameFinal,sizeFrameFinal);
            frameFinal[k] = ESC ;
            frameFinal[k + 1] = FLAGFLAGESC;
            k += 2;
        }else{
            frameFinal[k] = msg[i];
            k++;
        }
    }

    //calculo BCC2
    unsigned char BCC2 = msg[0];
    for (int j = 1; j < size; j++){
        BCC2 ^= msg[j];
    }
```

```c
//calculo BCC2
unsigned char BCC2 = msg[0];
for (int j = 1; j < size; j++){
    BCC2 ^= msg[j];
}



//Stuffing BCC2
if(BCC2 == FLAG){
    sizeFrameFinal++;
    frameFinal[k] = ESC;
    frameFinal[k + 1] = FLAGESC;
    k += 2;
}else if(BCC2 == ESC){
    sizeFrameFinal++;
    frameFinal[k] = ESC;
    frameFinal[k + 1] = FLAGFLAGESC;
    k += 2;
}else{
    frameFinal[k] = BCC2;
}

frameFinal[k + 1] = FLAG;
unsigned char* frameError;
//enviar FrameFinal
int stop = FALSE;
do {

    frameError = errorBCC1(frameFinal,sizeFrameFinal);//Altera BCC1 caso percentagem de erro > 0
    frameError = errorBCC2(frameFinal,sizeFrameFinal);//Altera BCC2 caso percentagem de erro > 0
    write(fd,frameError,sizeFrameFinal);

    //iniciar Alarm
    alarm(TIMEOUT);
    alarmEnabled = TRUE;
    //ler ControlMessage
    unsigned char controlWordReceived;
    receiveControlWord(fd, &controlWordReceived);

    if((controlWordReceived == RR0 && currentFrame == 1) || (controlWordReceived == RR1 && currentFrame == 0)){
        currentFrame ^= 1;
        alarmCount = 0;
        alarmEnabled = FALSE;
        stop = TRUE;
        free(frameError);
        return sizeFrameFinal;
    }else if(controlWordReceived == REJ0 || controlWordReceived == REJ1 ){
        printf("REJ received\n");
        if(controlWordReceived == REJ0 && currentFrame == 1){
            currentFrame = 0;
            break;
        }else if(controlWordReceived == REJ1 && currentFrame == 0){
            currentFrame = 1;
            break;
        }
        alarmCount = 0;
        alarm(0);
    }
    controlWordReceived = 0xFF;
```

```
    do {

        frameError = errorBCC1(frameFinal,sizeFrameFinal);//Altera BCC1 caso percentagem de erro > 0
        frameError = errorBCC2(frameFinal,sizeFrameFinal);//Altera BCC2 caso percentagem de erro > 0
        write(fd,frameError,sizeFrameFinal);

        //iniciar Alarm
        alarm(TIMEOUT);
        alarmEnabled = TRUE;
        //ler ControlMessage
        unsigned char controlWordReceived;
        receiveControlWord(fd, &controlWordReceived);

        if((controlWordReceived == RR0 && currentFrame == 1) || (controlWordReceived == RR1 && currentFrame == 0)){
            currentFrame ^= 1;
            alarmCount = 0;
            alarmEnabled = FALSE;
            stop = TRUE;
            free(frameError);
            return sizeFrameFinal;
        }else if(controlWordReceived == REJ0 || controlWordReceived == REJ1 ){
            printf("REJ received\n");
            if(controlWordReceived == REJ0 && currentFrame == 1){
                currentFrame = 0;
                break;
            }else if(controlWordReceived == REJ1 && currentFrame == 0){
                currentFrame = 1;
                break;
            }
            alarmCount = 0;
            alarm(0);
        }
        controlWordReceived = 0xFF;

    }while(!stop && alarmCount < MAX_SENDS);

    free(frameError);

    if(alarmCount >= MAX_SENDS){
        printf("time-out max sends exceeded\n");
        exit(-1);
    }
    return -1;
}
```

```c
int LLCLOSE(int fd){
    unsigned char charReceived;
    printf("LLCLOSE STARTED\n");
    do{
        sendControlWord(fd, DISC);
        alarm(TIMEOUT);
        alarmEnabled = TRUE;
        printf("DISC SENT\n");
        receiveControlWord(fd,&charReceived);
        if(charReceived == DISC){
            printf("DISC RECEIVED\n");
            alarmCount = 0;
            alarm(0);
        }else{
            charReceived = 0xFF;
        }
    }while((charReceived != DISC) && (alarmCount < MAX_SENDS) );

    if(alarmCount >= MAX_SENDS){
        exit(-1);
    }
    printf("UA SENT\n");
    sendControlWord(fd,UA);

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

    close(fd);
}

int LLOPEN(int fd){

     if (tcgetattr(fd, &oldtio) == -1)
    {
        perror("tcgetattr");
        exit(-1);
    }

    memset(&newtio, 0, sizeof(newtio));

    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;

    newtio.c_lflag = 0;
    newtio.c_cc[VTIME] = 1; // 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);
    }

    printf("New termios structure set\n");
```

```
    unsigned char c;
    do {
        sendControlWord(fd,SET);
        printf("SET SENT\n");
        alarm(TIMEOUT);
        alarmEnabled = TRUE;

        unsigned char cReceived;
        receiveControlWord(fd,&cReceived);
        if(cReceived == UA){
            printf("UA received\n");
            UAreceived = TRUE;
            alarmCount = 0;
            alarm(0);
        }
    }while(!UAreceived && alarmCount < MAX_SENDS);

    if(alarmCount >= MAX_SENDS){
        return FALSE;
    }else{
        return TRUE;
    }
}

unsigned char* openFile(unsigned char *fileName, int* sizeFile){
    FILE * f;
    unsigned char* fileBytes;

    f = fopen((char*)fileName,"rb");
    if(f == NULL){
        perror("Error opening file\n");
        exit(-1);
    }

    fseek(f,0,SEEK_END);

    (*sizeFile) = ftell(f);

    fseek(f,0,SEEK_SET);

    fileBytes = (unsigned char*)malloc(*sizeFile * sizeof(unsigned char));
    fread(fileBytes,sizeof(unsigned char),*sizeFile,f);

    return fileBytes;
}
```

```c
//cria packet com tamanho do ficheiro
unsigned char *createControlPacket(int start,int fileSize,int *sizeControlPacket){
    unsigned char* controlPacket = (unsigned char*)malloc(sizeof(unsigned char) * 7);

    if(start){
        controlPacket[0] = CP_START;
    }else{
        controlPacket[0] = CP_END;
    }


    controlPacket[1] = CP_TYPE_FILESIZE; //0
    controlPacket[2] = CP_LENGTH_FILESIZE; //4
    controlPacket[3] = (fileSize >> 24) & 0xFF;
    controlPacket[4] = (fileSize >> 16) & 0xFF;
    controlPacket[5] = (fileSize >> 8) & 0xFF;
    controlPacket[6] = fileSize & 0xFF;

    *sizeControlPacket = (int)(sizeof(unsigned char) * 7);

    return controlPacket;
}

unsigned char* addHeaderPacket(unsigned char* packet, int fileSize, int packetSize ){
    unsigned char* packetToSend;

    packetToSend = (unsigned char*)malloc(packetSize + 4);
    packetToSend[0] = Control_DATA_PACKET;
    packetToSend[1] = sequenceNumberPacket % 255;
    packetToSend[2] = fileSize / 256;
    packetToSend[3] = fileSize % 256;
    memcpy(packetToSend + 4,packet,packetSize);
    sequenceNumberPacket++;
    return packetToSend;
}

void createPacket(unsigned char* fileData,int*indexFile, int fileSize,int* packetSize, unsigned char* packet){

    if(*indexFile + *packetSize > fileSize){
        *packetSize = fileSize  - *indexFile;
    }
    packet = (unsigned char*)realloc(packet,*packetSize * sizeof(unsigned char));

    for(int i = 0; i < *packetSize;i++){
        packet[i] = fileData[*indexFile];
        (*indexFile)++;
    }

}
```

```c
int main(int argc, char *argv[])
{
    const char *serialPortName = argv[1];

    if (argc < 3)
    {
        printf("Incorrect program usage\n"
               "Usage: %s <SerialPort> <FileName\n"
               "Example: %s /dev/ttyS1 pinguim.gif\n",
               argv[0],
               argv[0]);
        exit(1);
    }

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


    int fileSize;
    unsigned char* fileBytes = openFile((unsigned char*)argv[2], &fileSize);

    struct timespec startTime, endTime;
    clock_gettime(CLOCK_REALTIME, &startTime);


    if(!LLOPEN(fd)){
        printf("time-out LLOPEN\n");
        exit(-1);
    }

    srand(time(NULL));
    int sizeControlPacket;

    unsigned char * startPacket = createControlPacket(TRUE, fileSize,&sizeControlPacket);
    LLWRITE(fd,startPacket,sizeControlPacket);
    printf("START PACKET SENT\n");

    int indexFile = 0;
    int packetSize = PACKET_SIZE;
    int numPackets = 0;
    int numPackage = 0;
    while(indexFile < fileSize && packetSize == PACKET_SIZE){
        unsigned char* packet = (unsigned char*) malloc(1 * sizeof(unsigned char));
        createPacket(fileBytes,&indexFile,fileSize,&packetSize, packet);

        unsigned char* packetToSend = addHeaderPacket(packet,fileSize,packetSize);
        int packetToSendSize = packetSize + 4;

        LLWRITE(fd,packetToSend,packetToSendSize);
        printf("Packet sent\n");
        free(packetToSend);
        free(packet);
    }
    free(fileBytes);
    unsigned char * endPacket = createControlPacket(FALSE, fileSize,&sizeControlPacket);
```

```
    unsigned char * endPacket = createControlPacket(FALSE, fileSize,&sizeControlPacket);
    LLWRITE(fd,endPacket,sizeControlPacket);
    printf("END PACKET SENT\n");


    LLCLOSE(fd);

    clock_gettime(CLOCK_REALTIME, &endTime);

    double elapsed = (endTime.tv_sec - startTime.tv_sec) + (endTime.tv_nsec - startTime.tv_nsec) / 1E9;

    //printf("Tempo:%f\n",elapsed);

    return 0;
}

unsigned char *errorBCC1(unsigned char* packet, int packetSize){
    unsigned char* packetError = (unsigned char*)malloc(packetSize *sizeof(unsigned char));
    memcpy(packetError,packet,packetSize);
    int prob = (rand() % 100) + 1;
    if(prob <= BCC1ERROR){
        int i = (rand() % 2);
        unsigned char randomLetter = (unsigned char)('A' + (rand() % 26));
        packetError[i] = randomLetter;
        printf("Error in BCC1\n");
    }
    return packetError;
}

unsigned char *errorBCC2(unsigned char* packet, int packetSize){
    unsigned char* packetError = (unsigned char*)malloc(packetSize *sizeof(unsigned char));
    memcpy(packetError,packet,packetSize);
    int prob = (rand() % 100) + 1;
    if(prob <= BCC2ERROR){
        int i = (rand() % (packetSize - 5)) + 4;
        unsigned char randomLetter = (unsigned char)('A' + (rand() % 26));
        packetError[i] = randomLetter;
        printf("Error in BCC2\n");
    }
    return packetError;
}
```

Receiver.h

```c
#ifndef REC_HEADER
#define REC_HEADER

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termios.h>
#include <unistd.h>

#define BAUDRATE B38400
#define _POSIX_SOURCE 1 // POSIX compliant source

#define FALSE 0
#define TRUE 1

#define FLAG (0x7E)
#define A (0x03)
#define A_1 (0x01)
#define SET (0x03)
#define UA (0x07)
#define DISC (0x0B)
#define C0 (0x00)
#define C1 (0x40)
#define ESC (0x7D)
#define FLAGESC (0x5E)
#define FLAGFLAGESC (0x5D)
#define RR0 (0x05)
#define RR1 (0x85)
#define REJ0 (0x01)
#define REJ1 (0x81)
#define CP_END (0x03)

//--------------------Data link layer--------------------------------

//Função com máquina de estados para leitura de controlWord
int receiveControlWord(int fd, unsigned char C);

//Função que verifica o BCC2
int verifyBCC2(unsigned char*messageReceived, int sizeMessageReceived);

//Função que cria uma trama de supervisão e envia
void sendControlWord(int fd, unsigned char C);

//Função de leitura de tramas
int LLREAD(int fd, unsigned char * messageReceived);

//Função de terminação da ligação
void LLCLOSE(int fd);

//Função de estabelecimento da ligação
int LLOPEN(int fd);
```

```
//--------------------Data link layer-------------------------------

//Função com máquina de estados para leitura de controlWord
int receiveControlWord(int fd, unsigned char C);

//Função que verifica o BCC2
int verifyBCC2(unsigned char*messageReceived, int sizeMessageReceived);

//Função que cria uma trama de supervisão e envia
void sendControlWord(int fd, unsigned char C);

//Função de leitura de tramas
int LLREAD(int fd, unsigned char * messageReceived);

//Função de terminação da ligação
void LLCLOSE(int fd);

//Função de estabelecimento da ligação
int LLOPEN(int fd);

//--------------------Application Layer---------------------------

//Função que verifica se um packet é o packet de terminação
int isEndPacket(unsigned char* packetReceived,int sizePacketReceived, unsigned char* startPacket, int sizeStartPacket);

//Função que remove o Header de controlo da application layer
unsigned char * removeControlHeader(unsigned char *packetReceived,int sizePacketReceived);

//Função que cria um ficheiro a partir dos dados recebidos
void createFile(unsigned char* fileData, int fileSize);

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

#endif //REC_HEADER
```

Receiver.c

```c
#include "receiver.h"

volatile int STOP = FALSE;
int expectedFrame = 0;
struct termios oldtio;
struct termios newtio;

int receiveControlWord(int fd, unsigned char C){
    int state = 0;
    unsigned char c = 0xFF;
    unsigned char A_check;
    while(state != 5){
        read(fd, &c,1);
        switch(state){
            case 0:
                if(c == FLAG){
                    state = 1;
                }
                break;
            case 1:
                if(c == A || c == A_1){
                    state = 2;
                }else{
                    if (c == FLAG){
                        state = 1;
                    }else{
                        state = 0;
                    }
                }
                break;
            case 2:
                if(c == C){
                    state = 3;
                }else{
                    if(c == FLAG)
                        state = 1;
                    else
                        state = 0;
                }
                break;
            case 3:
                if(C == DISC || C == UA){
                    A_check = A_1;
                }else{
                    A_check = A;
                }
                if(c == (A_check ^ C))
                    state = 4;
                else
                    state = 0;
                break;
            case 4:
                if(c == FLAG){
                    state = 5;
                }
                else
                    state = 0;
                break;
        }
    }
    return TRUE;
}
```

```
int verifyBCC2(unsigned char*messageReceived, int sizeMessageReceived){

    unsigned char BCC2 = messageReceived[0];
    for (int j = 1; j < sizeMessageReceived - 1; j++){
        BCC2 ^= messageReceived[j];
    }


    if(BCC2 == messageReceived[sizeMessageReceived - 1]){
        return TRUE;
    }else{
        return FALSE;
    }
}


void sendControlWord(int fd, unsigned char C){
    unsigned char message[5];
    message[0] = FLAG;
    if(C == DISC){
        message[1] = A_1;
    }else{
        message[1] = A;
    }
    message[2] = C;
    message[3] = message[1] ^ message[2];
    message[4] = FLAG;
    write(fd,message,5);
}

int LLREAD(int fd, unsigned char * messageReceived){
    int sizeMessageReceived = 0;

    int acceptedFrame =  FALSE;
    int receivedFrame;

    int state = 0;
    unsigned char readChar,saveC;
    int i= 0;
    while(state != 7){
        read(fd,&readChar,1);
        switch(state){
            case 0:
                if(readChar == FLAG){
                    state = 1;
                }else{
                    state = 0;
                }
                break;
            case 1:
                if(readChar == A){
                    state = 2;
                }else if(readChar == FLAG){
                    state = 1;
                }else{
                    state = 0;
                }
                break;
            case 2:
                if(readChar == C0 || readChar == C1){
                    state = 3;
```

```
            case 2:
                if(readChar == C0 || readChar == C1){
                    state = 3;
                    saveC = readChar;
                    if(readChar == C0){
                        receivedFrame = 0;
                    }else{
                        receivedFrame = 1;
                    }
                }else if(readChar == FLAG){
                    state = 1;
                }else{
                    state = 0;
                }
                break;
            case 3:
                if(readChar == (A^saveC)){
                    state = 4;
                }else{
                    printf("error in the protocol\n");
                    state = 0;
                }
                break;
            case 4:
                if(readChar == FLAG){
                    state = 6;
                }else if(readChar == ESC){
                    state = 5;
                }else{
                    sizeMessageReceived++;
                    messageReceived = (unsigned char*)realloc(messageReceived, sizeMessageReceived);
                    messageReceived[sizeMessageReceived - 1 ] = readChar;
                }
                break;
            case 5:
                if(readChar == FLAGESC){
                    //destuffing trocar ESC e FLAGESC por FLAG que é igual a adicionar a FLAG à msg
                    sizeMessageReceived++;
                    messageReceived = (unsigned char*)realloc(messageReceived, sizeMessageReceived);
                    messageReceived[sizeMessageReceived - 1 ] = FLAG;
                }else if(readChar == FLAGFLAGESC){
                    //destuffing trocar ESC e FLAGFLAGESC por ESC que é igual a adicionar a ESC à msg
                    sizeMessageReceived++;
                    messageReceived = (unsigned char*)realloc(messageReceived, sizeMessageReceived);
                    messageReceived[sizeMessageReceived - 1 ] = ESC;
                }
                state = 4;
                break;
            case 6:
                if(receivedFrame == expectedFrame){
                    if(verifyBCC2(messageReceived,sizeMessageReceived)){
                        if(receivedFrame == 0){
                            sendControlWord(fd,RR1);
                            state = 7;
                        }else if(receivedFrame == 1){
                            sendControlWord(fd,RR0);
                            state = 7;
                        }
                        acceptedFrame = TRUE;
                    }else{
```

```
            case 6:
                if(receivedFrame == expectedFrame){
                    if(verifyBCC2(messageReceived,sizeMessageReceived)){
                        if(receivedFrame == 0){
                            sendControlWord(fd,RR1);
                            state = 7;
                        }else if(receivedFrame == 1){
                            sendControlWord(fd,RR0);
                            state = 7;
                        }
                        acceptedFrame = TRUE;
                    }else{
                        if(receivedFrame == 0){
                            sendControlWord(fd,REJ0);
                            state = 7;
                        }else{
                            sendControlWord(fd,REJ1);
                            state = 7;
                        }
                        acceptedFrame = FALSE;
                        printf("error in the data\n");
                    }
                }else{
                    //Frame recebido repetido
                    printf("duplicate frame\n");
                    if(receivedFrame == 0){
                        sendControlWord(fd,REJ1);
                        state = 7;
                    }else{
                        sendControlWord(fd,REJ0);
                        state = 7;
                    }
                    acceptedFrame = FALSE;
                }

                break;
        }
    }

    //retirar BCC2 da mensagem
    messageReceived = (unsigned char*)realloc(messageReceived,sizeMessageReceived - 1);
    sizeMessageReceived--;

    int sizeReturn;

    if(acceptedFrame){
        if(receivedFrame == expectedFrame){
            expectedFrame ^= 1;

            sizeReturn = sizeMessageReceived;
        }else{
            sizeReturn = -1;
        }
    }else{
        sizeReturn = -1;
    }

    return sizeReturn;
}
```

```c
void LLCLOSE(int fd){
    printf("LLCLOSE STARTED\n");
    receiveControlWord(fd,DISC);
    printf("DISC RECEIVED\n");
    sendControlWord(fd,DISC);
    printf("DISC SENT\n");


    if (tcsetattr(fd, TCSANOW, &oldtio) == -1)
    {
        perror("tcsetattr");
        exit(-1);
    }
    close(fd);
}


int LLOPEN(int fd){

    if (tcgetattr(fd, &oldtio) == -1)
    {
        perror("tcgetattr");
        exit(-1);
    }

    memset(&newtio, 0, sizeof(newtio));

    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;

    newtio.c_lflag = 0;
    newtio.c_cc[VTIME] = 1; // 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);
    }

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


    if(receiveControlWord(fd,SET)){
        printf("SET RECEIVED\n");
        sendControlWord(fd,UA);
        printf("UA SENT\n");
        return TRUE;
    }

    return FALSE;

}
```

```c
int isEndPacket(unsigned char* packetReceived,int sizePacketReceived, unsigned char* startPacket, int sizeStartPacket){
    if(sizePacketReceived != sizeStartPacket){
        return FALSE;
    }
    if(packetReceived[0] != CP_END){
        return FALSE;
    }else{
        for(int i = 1; i < sizePacketReceived; i++){
            if(packetReceived[i] != startPacket[i]){
                return FALSE;
            }
        }
    }
    return TRUE;
}

unsigned char * removeControlHeader(unsigned char *packetReceived,int sizePacketReceived){
    unsigned char* newPacketWithoutHeader = (unsigned char*)malloc(sizePacketReceived - 4);
    int j = 4;
    for(int i = 0; i < (sizePacketReceived - 4);i++){
        newPacketWithoutHeader[i] = packetReceived[j];
        j++;
    }
    return newPacketWithoutHeader;
}

void createFile(unsigned char* fileData, int fileSize){
    FILE *fp;
    fp = fopen("pinguimReceiver.gif","wb+");
    fwrite((void*)fileData,1,fileSize,fp);
    fclose(fp);
}
```

```c
int main(int argc, char *argv[])
{
    const char *serialPortName = argv[1];

    if (argc < 2)
    {
        printf("Incorrect program usage\n"
               "Usage: %s <SerialPort>\n"
               "Example: %s /dev/ttyS1\n",
               argv[0],
               argv[0]);
        exit(1);
    }

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

    if(!LLOPEN(fd)){
        exit(-1);
    };

    unsigned char* startPacket = (unsigned char*)malloc(0);
    unsigned char* fileBytes ;
    int fileSize;
    int index = 0;
    int sizeStartPacket;

    sizeStartPacket = LLREAD(fd,startPacket);
    if(sizeStartPacket > 0 ){
        fileSize = (startPacket[3] << 24) | (startPacket[4] << 16) | (startPacket[5] << 8) | startPacket[6];
        printf("FILE HAS %i bytes\n",fileSize);
    }


    fileBytes = (unsigned char*)malloc(fileSize * sizeof(unsigned char));

    unsigned char * packetReceived;
    int sizePacketReceived;
    while(TRUE){
        packetReceived = (unsigned char*)malloc(0);
        sizePacketReceived = LLREAD(fd,packetReceived);
        if(sizePacketReceived > 0){
            printf("Packet received\n");
            if(isEndPacket(packetReceived,sizePacketReceived,startPacket,sizeStartPacket)){
                printf("End Packect received\n");
                break;
            }else{
                packetReceived = removeControlHeader(packetReceived,sizePacketReceived);
                memcpy(fileBytes + index,packetReceived,sizePacketReceived - 4);
                index += (sizePacketReceived - 4);
            }
        }
        free(packetReceived);

    }
```

```
    while(TRUE){
        packetReceived = (unsigned char*)malloc(0);
        sizePacketReceived = LLREAD(fd,packetReceived);
        if(sizePacketReceived > 0){
            printf("Packet received\n");
            if(isEndPacket(packetReceived,sizePacketReceived,startPacket,sizeStartPacket)){
                printf("End Packect received\n");
                break;
            }else{
                packetReceived = removeControlHeader(packetReceived,sizePacketReceived);
                memcpy(fileBytes + index,packetReceived,sizePacketReceived - 4);
                index += (sizePacketReceived - 4);
            }
        }
        free(packetReceived);

    }

    createFile(fileBytes,fileSize);


    LLCLOSE(fd);

    return 0;
}
```

```c
#include "receiver.h"

volatile int STOP = FALSE;
int expectedFrame = 0;
struct termios oldtio;
struct termios newtio;

int receiveControlWord(int fd, unsigned char C){
    int state = 0;
    unsigned char c = 0xFF;
    unsigned char A_check;
    while(state != 5){
        read(fd, &c,1);
        switch(state){
            case 0:
                if(c == FLAG){
                    state = 1;
                }
                break;
            case 1:
                if(c == A || c == A_1){
                    state = 2;
                }else{
                    if (c == FLAG){
                        state = 1;
                    }else{
                        state = 0;
                    }
                }
                break;
            case 2:
                if(c == C){
                    state = 3;
                }else{
                    if(c == FLAG)
                        state = 1;
                    else
                        state = 0;
                }
                break;
            case 3:
                if(C == DISC || C == UA){
                    A_check = A_1;
                }else{
                    A_check = A;
                }
                if(c == (A_check ^ C))
                    state = 4;
                else
                    state = 0;
                break;
            case 4:
                if(c == FLAG){
                    state = 5;
                }
                else
                    state = 0;
                break;
        }
    }
    return TRUE;
}
```

```
int verifyBCC2(unsigned char*messageReceived, int sizeMessageReceived){

    unsigned char BCC2 = messageReceived[0];
    for (int j = 1; j < sizeMessageReceived - 1; j++){
        BCC2 ^= messageReceived[j];
    }


    if(BCC2 == messageReceived[sizeMessageReceived - 1]){
        return TRUE;
    }else{
        return FALSE;
    }
}


void sendControlWord(int fd, unsigned char C){
    unsigned char message[5];
    message[0] = FLAG;
    if(C == DISC){
        message[1] = A_1;
    }else{
        message[1] = A;
    }
    message[2] = C;
    message[3] = message[1] ^ message[2];
    message[4] = FLAG;
    write(fd,message,5);
}

int LLREAD(int fd, unsigned char * messageReceived){
    int sizeMessageReceived = 0;

    int acceptedFrame =  FALSE;
    int receivedFrame;

    int state = 0;
    unsigned char readChar,saveC;
    int i= 0;
    while(state != 7){
        read(fd,&readChar,1);
        switch(state){
            case 0:
                if(readChar == FLAG){
                    state = 1;
                }else{
                    state = 0;
                }
                break;
            case 1:
                if(readChar == A){
                    state = 2;
                }else if(readChar == FLAG){
                    state = 1;
                }else{
                    state = 0;
                }
                break;
            case 2:
                if(readChar == C0 || readChar == C1){
                    state = 3;
```

```
            case 2:
                if(readChar == C0 || readChar == C1){
                    state = 3;
                    saveC = readChar;
                    if(readChar == C0){
                        receivedFrame = 0;
                    }else{
                        receivedFrame = 1;
                    }
                }else if(readChar == FLAG){
                    state = 1;
                }else{
                    state = 0;
                }
                break;
            case 3:
                if(readChar == (A^saveC)){
                    state = 4;
                }else{
                    printf("error in the protocol\n");
                    state = 0;
                }
                break;
            case 4:
                if(readChar == FLAG){
                    state = 6;
                }else if(readChar == ESC){
                    state = 5;
                }else{
                    sizeMessageReceived++;
                    messageReceived = (unsigned char*)realloc(messageReceived, sizeMessageReceived);
                    messageReceived[sizeMessageReceived - 1 ] = readChar;
                }
                break;
            case 5:
                if(readChar == FLAGESC){
                    //destuffing trocar ESC e FLAGESC por FLAG que é igual a adicionar a FLAG à msg
                    sizeMessageReceived++;
                    messageReceived = (unsigned char*)realloc(messageReceived, sizeMessageReceived);
                    messageReceived[sizeMessageReceived - 1 ] = FLAG;
                }else if(readChar == FLAGFLAGESC){
                    //destuffing trocar ESC e FLAGFLAGESC por ESC que é igual a adicionar a ESC à msg
                    sizeMessageReceived++;
                    messageReceived = (unsigned char*)realloc(messageReceived, sizeMessageReceived);
                    messageReceived[sizeMessageReceived - 1 ] = ESC;
                }
                state = 4;
                break;
            case 6:
                if(receivedFrame == expectedFrame){
                    if(verifyBCC2(messageReceived,sizeMessageReceived)){
                        if(receivedFrame == 0){
                            sendControlWord(fd,RR1);
                            state = 7;
                        }else if(receivedFrame == 1){
                            sendControlWord(fd,RR0);
                            state = 7;
                        }
                        acceptedFrame = TRUE;
                    }else{
```

```
            case 6:
                if(receivedFrame == expectedFrame){
                    if(verifyBCC2(messageReceived,sizeMessageReceived)){
                        if(receivedFrame == 0){
                            sendControlWord(fd,RR1);
                            state = 7;
                        }else if(receivedFrame == 1){
                            sendControlWord(fd,RR0);
                            state = 7;
                        }
                        acceptedFrame = TRUE;
                    }else{
                        if(receivedFrame == 0){
                            sendControlWord(fd,REJ0);
                            state = 7;
                        }else{
                            sendControlWord(fd,REJ1);
                            state = 7;
                        }
                        acceptedFrame = FALSE;
                        printf("error in the data\n");
                    }
                }else{
                    //Frame recebido repetido
                    printf("duplicate frame\n");
                    if(receivedFrame == 0){
                        sendControlWord(fd,REJ1);
                        state = 7;
                    }else{
                        sendControlWord(fd,REJ0);
                        state = 7;
                    }
                    acceptedFrame = FALSE;
                }

                break;
        }
    }

    //retirar BCC2 da mensagem
    messageReceived = (unsigned char*)realloc(messageReceived,sizeMessageReceived - 1);
    sizeMessageReceived--;

    int sizeReturn;

    if(acceptedFrame){
        if(receivedFrame == expectedFrame){
            expectedFrame ^= 1;

            sizeReturn = sizeMessageReceived;
        }else{
            sizeReturn = -1;
        }
    }else{
        sizeReturn = -1;
    }

    return sizeReturn;
}
```

```c
void LLCLOSE(int fd){
    printf("LLCLOSE STARTED\n");
    receiveControlWord(fd,DISC);
    printf("DISC RECEIVED\n");
    sendControlWord(fd,DISC);
    printf("DISC SENT\n");


    if (tcsetattr(fd, TCSANOW, &oldtio) == -1)
    {
        perror("tcsetattr");
        exit(-1);
    }
    close(fd);
}


int LLOPEN(int fd){

    if (tcgetattr(fd, &oldtio) == -1)
    {
        perror("tcgetattr");
        exit(-1);
    }

    memset(&newtio, 0, sizeof(newtio));

    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;

    newtio.c_lflag = 0;
    newtio.c_cc[VTIME] = 1; // 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);
    }

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


    if(receiveControlWord(fd,SET)){
        printf("SET RECEIVED\n");
        sendControlWord(fd,UA);
        printf("UA SENT\n");
        return TRUE;
    }

    return FALSE;

}
```

```
int isEndPacket(unsigned char* packetReceived,int sizePacketReceived, unsigned char* startPacket, int sizeStartPacket){
    if(sizePacketReceived != sizeStartPacket){
        return FALSE;
    }
    if(packetReceived[0] != CP_END){
        return FALSE;
    }else{
        for(int i = 1; i < sizePacketReceived; i++){
            if(packetReceived[i] != startPacket[i]){
                return FALSE;
            }
        }
    }
    return TRUE;
}

unsigned char * removeControlHeader(unsigned char *packetReceived,int sizePacketReceived){
    unsigned char* newPacketWithoutHeader = (unsigned char*)malloc(sizePacketReceived - 4);
    int j = 4;
    for(int i = 0; i < (sizePacketReceived - 4);i++){
        newPacketWithoutHeader[i] = packetReceived[j];
        j++;
    }
    return newPacketWithoutHeader;
}

void createFile(unsigned char* fileData, int fileSize){
    FILE *fp;
    fp = fopen("pinguimReceiver.gif","wb+");
    fwrite((void*)fileData,1,fileSize,fp);
    fclose(fp);
}
```

```c
int main(int argc, char *argv[])
{
    const char *serialPortName = argv[1];

    if (argc < 2)
    {
        printf("Incorrect program usage\n"
               "Usage: %s <SerialPort>\n"
               "Example: %s /dev/ttyS1\n",
               argv[0],
               argv[0]);
        exit(1);
    }

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

    if(!LLOPEN(fd)){
        exit(-1);
    };

    unsigned char* startPacket = (unsigned char*)malloc(0);
    unsigned char* fileBytes ;
    int fileSize;
    int index = 0;
    int sizeStartPacket;

    sizeStartPacket = LLREAD(fd,startPacket);
    if(sizeStartPacket > 0 ){
        fileSize = (startPacket[3] << 24) | (startPacket[4] << 16) | (startPacket[5] << 8) | startPacket[6];
        printf("FILE HAS %i bytes\n",fileSize);
    }


    fileBytes = (unsigned char*)malloc(fileSize * sizeof(unsigned char));

    unsigned char * packetReceived;
    int sizePacketReceived;
    while(TRUE){
        packetReceived = (unsigned char*)malloc(0);
        sizePacketReceived = LLREAD(fd,packetReceived);
        if(sizePacketReceived > 0){
            printf("Packet received\n");
            if(isEndPacket(packetReceived,sizePacketReceived,startPacket,sizeStartPacket)){
                printf("End Packect received\n");
                break;
            }else{
                packetReceived = removeControlHeader(packetReceived,sizePacketReceived);
                memcpy(fileBytes + index,packetReceived,sizePacketReceived - 4);
                index += (sizePacketReceived - 4);
            }
        }
        free(packetReceived);

    }
```

```
    while(TRUE){
        packetReceived = (unsigned char*)malloc(0);
        sizePacketReceived = LLREAD(fd,packetReceived);
        if(sizePacketReceived > 0){
            printf("Packet received\n");
            if(isEndPacket(packetReceived,sizePacketReceived,startPacket,sizeStartPacket)){
                printf("End Packect received\n");
                break;
            }else{
                packetReceived = removeControlHeader(packetReceived,sizePacketReceived);
                memcpy(fileBytes + index,packetReceived,sizePacketReceived - 4);
                index += (sizePacketReceived - 4);
            }
        }
        free(packetReceived);

    }

    createFile(fileBytes,fileSize);


    LLCLOSE(fd);

    return 0;
}
```