Universidade do Porto
FEUP Faculdade de Engenharia

# Relatório Redes de Computadores

David Baião - up201305195
Filipa Ramos - up201305378
Inês Carneiro - up201303501

## Índice



## Sumário

No âmbito da unidade curricular de redes de computadores foi desenvolvido um programa em C para configurar a ligação entre dois computadores através da porta de série permitindo o envio de ficheiros entre ambos (o exemplo fornecido era o de uma imagem de um pinguim).

Serve o presente relatório o propósito de exemplificar a estruturação do código, as interfaces e as principais estruturas usadas. É ainda feita a descrição das camadas de ligação e de aplicação.

## 1. Introdução

O problema introduzido no âmbito da unidade curricular de redes de computadores era o seguinte: estabelecer um protocolo de ligação de dados entre dois computadores através da porta de série e testá-lo com recurso à transferência de um ficheiro entre ambos.

O presente relatório serve o propósito de explicitar a implementação escolhida para resolver o problema proposto. O ambiente de desenvolvimento foi um computador com LINUX, a linguagem usada foi C e as portas de série eram RS-232 com comunicação assíncrona. No mesmo, é explicada tanto a arquitetura da aplicação desenvolvida como a estrutura da mesma (secções 2 e 3 respetivamente). Para além disto, são apresentados casos de uso principais na secção 4. As secções 5 e 6 aprofundam as camadas de ligação e aplicação respetivamente. Na penúltima secção são referidos os testes efetuados para validar a aplicação e avaliar a robustez da mesma. A última secção, de número 8, é referente aos elementos de valorização que foram implementados.

## 2. Arquitetura

O programa está dividido em duas partes, o protocolo (*link layer*) e a aplicação (*application layer*).

A *application layer* serve para o tratamento do ficheiro para envio no caso do transmissor e da receção no caso do recetor, ou seja, no caso do transmissor, o programa abre o ficheiro especificado, calcula o valor de tramas que devem ser enviadas e usa o *link layer* para que o recetor receba com segurança as tramas corretas. Já no receiver, além de

receber as tramas, trata de extrair os dados necessários e escreve um ficheiro com esses valores de maneira a produzir uma cópia do ficheiro enviado pelo emissor.

O *link layer* possui as funções usadas pela *application layer* para envio e receção de tramas, bem como as funções auxiliares de tratamento das tramas.

## 3. Estrutura do Código

### appplicationlayer.c

A *application layer* possui a função *main* do programa, bem como as funções que vão tratar os dados do ficheiro para envio e a receção destes dados pelo outro computador.

Em applicationlayer.h encontra-se a *struct* usada por esta camada.

```
struct applicationLayer {
    int fd; // descritor de ficheiro
    int flag; /*TRANSMITTER | RECEIVER*/

    char* filename; //file name
    int filesize; //file size
    int lengthDados;
    char* buf; //escrevemos sempre no mesmo buffer ele é sempre reescrito
    int numDataPack;
    unsigned char seqNumb;
    char * dados;
    char * porta;
};
```

Esta *struct* é usada principalmente para guardar valores referentes ao ficheiro a enviar/receber bem como outras informações importantes, tais como o descritor do ficheiro e uma *flag* que distingue o emissor do recetor.

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

O programa inicia nesta função. Verifica o valor dos argumentos dados e espera que para além do nome do executável apenas exista outro argumento com o endereço da porta a usar. Inicia a estrutura de dados a ser usada e apresenta o Menu inicial. Depois, consoante a opção escolhida, esta função continua o programa da forma pretendida.

**int app_layer_transmitter()**

Corresponde à função principal da *app_layer* por parte do transmissor. Esta função abre o ficheiro que se pretende enviar, calcula a quantidade de *packages* que deve ser enviado e usando as funções, tanto da *applicationLayer* como da *LinkLayer* envia todas as tramas necessárias.

**int app_layer_receiver()**

Esta função faz o mesmo que a anterior, mas de forma “inversa”, ou seja, em vez de enviar o ficheiro, esta função é usada pelo receptor para receber de forma correta o ficheiro.

**int makeCONTROLpackage(char* buf,int c)**

Função responsável pela criação de um pacote de controlo, pacotes estes que são enviados antes e após os dados.

**int makeDATApackage(char* buf,int seqNumb, int lengthDados, char* dados)**

Cria os pacotes que possuem os dados a serem enviados.

**char* processBuf(unsigned char seqnumb)**

Retira dos pacotes de dados os dados a serem guardados no ficheiro final. Retorna um apontador com os dados resultantes.

**int Settings(int trys, int timeO, int BR, int FrameSize)**

Altera o valor das definições usadas pelo programa. Permite alterar numero de tentativas, *timeout* , *baud rate* e por ultimo o tamanho da trama.

**link_layer.c**

O *link layer* possui as funções responsáveis pelo envio e receção de tramas e de manipulação e verificação destas.

Em link_layer.h encontra-se a *struct* de apoio às funções de ligação de dados.

```
struct Info {
  int fd; // descritor de ficheiro
  struct termios oldtio;
  struct termios newtio;
  char * endPorta; /*Dispositivo /dev/ttySx, x = 0, 1*/

  int baudRate; /*Velocidade de transmissão*/
  unsigned int sequenceNumber; /*Número de sequência da trama: 0, 1*/
  unsigned int timeout; /*Valor do temporizador: 1 s*/
  unsigned int numTransmissions; /*Número de tentativas em caso de falha*/

  int flag;

  char * dados; /*dados a enviar/receber*/
  int lengthDados;
  int tentativas;

  char * frameTemp; // serve para guardar uma frame temporariamente
  int frameTempLength;
  char * frameSend;
  int frameSendLength;

  int lostPack;

};
```

Esta *struct* é usada para guardar variáveis importantes como o descritor do ficheiro de transmissão ou relacionadas com o alarme, assim como para guardar tramas a serem enviadas ou que tenham sido recebidas.

**int llopen(char * porta, int flag);**
**int llwrite(int fd, char * buffer, int length);**
**int llread(int fd, char * buffer);**
**int llclose_transmitter(int fd);**
**int llclose_receiver(int fd);**

**int readFrame(char * frame)**

Lê uma trama para o parâmetro “frame” e retorna o tamanho da *frame* lida.

**char * verifyFrameType(char * frame)**

retorna o tipo da trama do argumento “frame” verificando o campo de controlo desta.

**int verifyFrame(char * frame, int length, char * type)**

Esta função é utilizada para verificar se uma trama está correta e é usada maioritariamente para verificar se a trama que foi recebida possui alguma irregularidade. Funciona para qualquer tipo de trama conhecida neste trabalho e usa uma máquina de estados para verificar se estas tramas estão corretas.

**int buildFrame(int flag, char * type)**

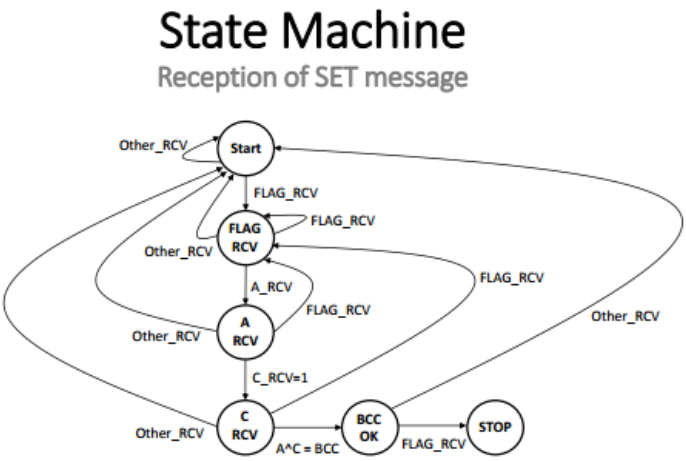
Cria uma trama de supervisão do tipo “type”.

**char * comporTramal(int flag, char * buffer, int length)**

Cria uma trama de informação com os dados presentes no argumento“buffer”.

**void state_machine(int state, char signal, char * type)**

*State machine* usada para verificar se uma trama está correta.




**void atende(int sig)**

Função chamada pelo *handler* do alarme, que quando chamada envia novamente a trama guardada.

**void stuffing(unsigned char* frame, unsigned int* size)**

Função responsável pelo *stuffing* de uma trama.

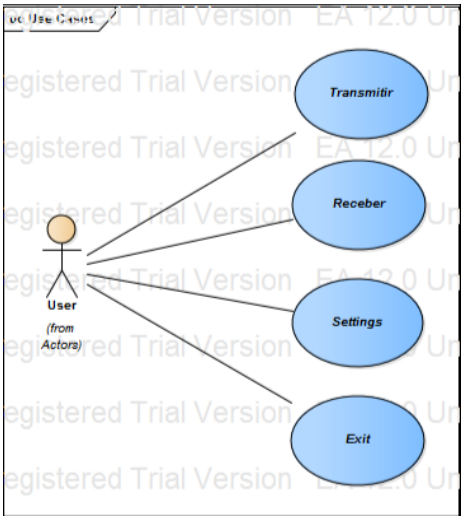
**void destuffing(unsigned char* frame, unsigned int* size)**

Função responsável pelo *destuffing* de uma trama.

## 4. Casos de uso principais

Para correr o programa deve ser feita a compilação dos ficheiros applicationlayer.c, link_layer.c e alarme.c, seguida da chamada do executável com apenas um argumento, sendo este correspondente ao endereço da porta a ser usada.

Quando o programa é executado, é apresentado um menu com as seguintes opções:




- Transmitir - Serve para transmitir um ficheiro. Após a seleção desta opção é perguntado ao utilizador o ficheiro que este quer enviar.
- Receber - Deve ser escolhido pelo computador que receberá o ficheiro.
- *Settings* - Menu responsável pela alteração das opções do programa, nomeadamente o número de tentativas, o tempo de espera do alarme, a *BaudRate* e o tamanho máximo das *frames* a enviar. A quando da alteração das definições, estas devem ser alteradas de igual modo em ambos os computadores.
- *Exit* - Para terminar o programa.

O computador que vai receber o ficheiro deve executar em primeiro lugar e esperar pelo transmissor. O transmissor, após dar o endereço do ficheiro a enviar inicia a *link_layer* e procede à transmissão do ficheiro.

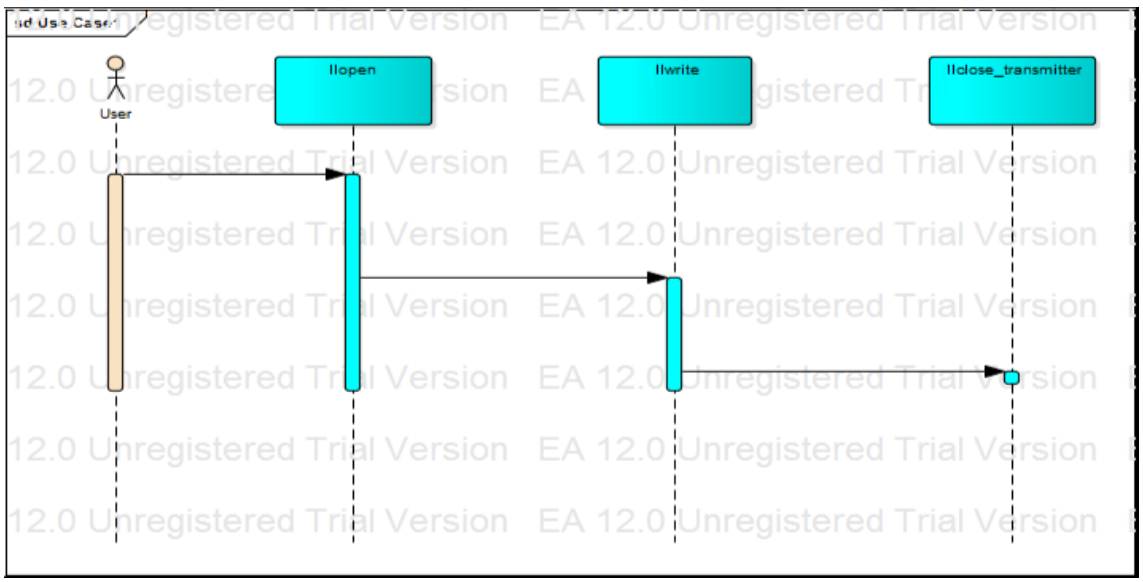



**Diagrama de sequência do transmissor**

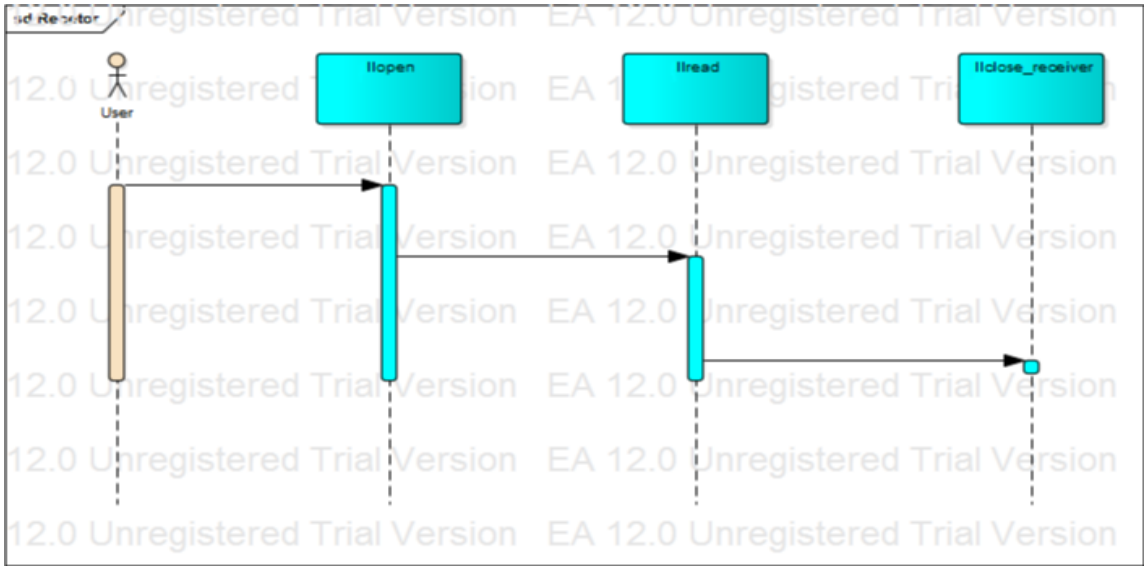



**Diagrama de sequência do recetor**

## 5. Protocolo de ligação lógica

**llopen**

A função *llopen* é responsável pela inicialização da *struct* da *link_layer* e pela abertura da porta para comunicar. De seguida o transmissor envia uma trama do tipo “set” e espera uma resposta do recetor, resposta esta correspondente a uma trama do tipo “ua”. Caso o transmissor não tiver recebido a trama “ua” corretamente dentro do tempo especificado o *handler* do alarme chama a função atende() que vai ser responsável por reenviar a trama “set”.

**llclose_transmitter e llclose_receiver**

Estas funções servem ambas para terminar a transmissão. A do transmissor envia uma trama do tipo “disc” e espera receber uma trama “disc” enviada pelo recetor para depois enviar uma “ua” e fechar a transmissão e a porta.

O recetor além das verificações normais, vê também se recebe uma trama de informação que possa ter faltado.

**llwrite**

O *llwrite* é responsável pelo envio de tramas de informação por parte do transmissor para o recetor. Após o envio da trama I, o transmissor deve esperar pela resposta do recetor. Caso não receba resposta ou esta não for a correta, a trama de informação é reenviada.

**llread**

A função *llread* recebe as tramas de informação vindas do transmissor e verificar se estão corretas. Se estiverem corretas é enviada uma trama do tipo “rr” para o transmissor, caso contrário envia uma trama do tipo “rej”.

Por vezes pode acontecer de o recetor terminar o *llopen* porque recebeu a trama “set” do transmissor e reenviou a trama “ua”, mas esta pode não chegar corretamente ao transmissor e portanto esta continua a enviar tramas “set”. Para corrigir esta situação

colocamos uma verificação no *llread* de maneira a caso este receba uma trama “set” reenvie uma trama “ua” para terminar o *llopen* do transmissor.

## 6. Protocolo de aplicação

O protocolo de aplicação envia dois tipos de pacotes:

**Pacotes de controlo**

Os pacotes de controlo são enviados no inicio e fim da transmissão e possuem informações sobre o ficheiro, tais como o nome e tamanho do ficheiro. A função usada para criar estes pacotes é a makeCONTROLpackage e a única diferença entre a inicial e a final é o primeiro carácter (1 para inicio e 2 para final).

**Pacotes de dados**

Os pacotes de dados possuem a informação do ficheiro a ser enviado e são enviados entre os dois pacotes de controlo.

## 7. Validação

Foram efetuados testes para os seguintes casos:

1. cabo retirado a meio da operação de transferência e voltado a ser ligado;
2. cabo retirado sem voltar a ligar;
3. receção do UA falhada;
4. receção do RR falhada;
5. falhas no envio do SET, DISC e o UA final;
6. erros no *stuffing* e *destuffing*;
7. cabo retirado e inserção de objetos pontiagudos nos *pins* do cabo, voltando a ligar o mesmo;
8. erros nas *flags* das *frames*;
9. erro na *flag* inicial que indica se é recetor ou transmissor;
10. erros nas definições;
11. variação do tamanho da frame;
12. variação do *baud rate*;
13. variação das tentativas;
14. variação do time out;
15. erros na finalização do programa quando ocorre time out;
16. funcionamento anormal do alarme;
17. número de argumentos errado.

Todos estes testes levaram a alterações no código que o tornaram mais eficiente e robusto. Devido ao elevado número de testes realizados, a ocorrência de *bugs* será mais difícil embora seja possível.

## 8. Elementos de Valorização

O programa contem um menu inicial que permitia escolher as definições que incluíam os valores de *baud rate*, número de retransmissões e tempo de time out.

```c
int changeSettings(){
    char fileC[20];
    int tentativasC;
    int timeOutC;
    int baudRateC;
    int max_size;

    printf("tentativas: ");
    scanf("%d", &tentativasC);

    printf("timeOut: ");
    scanf("%d", &timeOutC);

    printf("BaudRate: ");
    scanf("%d", &baudRateC);

    printf("Max_Frame_Size: ");
    scanf("%d", &max_size);

    Settings(tentativasC, timeOutC, baudRateC, max_size);
    return 1;

}
```

```c
int Settings(int trys, int timeO, int BR, int FrameSize){
    tentativas = trys;
    timeOut = timeO;
    BaudRate = convertBaudrate(BR);
    Max_Frame_Size = 2*FrameSize+2+4+8;

    return 1;
}
```

É ainda possível verificar na aplicação estatísticas sobre o número de pacotes perdidos e o tamanho do ficheiro recebido aquando da transmissão.

Foi utilizado para as verificações das tramas uma máquina de estados que serve para todo o tipo de tramas o que facilita imenso as verificações (ver Anexo II).

## 9. Conclusões

A transmissão de ficheiros entre dois computadores através da porta de série pode ser configurada através de 4 funções principais que permitem abrir a porta de série, fechá-la, ler dados e escrevê-los. Estas funções fazem parte da camada de ligação. Para além destas, esta camada contém funções para construir *frames*, verificá-las e organizá-las por tipos.

A outra camada, a de aplicação, serve para construir os packages tanto de controlo como de informação. Esta camada trata de chamar todas as outras funções, juntando os processos num só que forma a *thread* principal.

Este projeto revela-se extremamente útil para melhor compreender os processos de transferência de informação entre sistemas e para aprofundar o conhecimento da linguagem de programação C.

# Anexos

## Anexo I

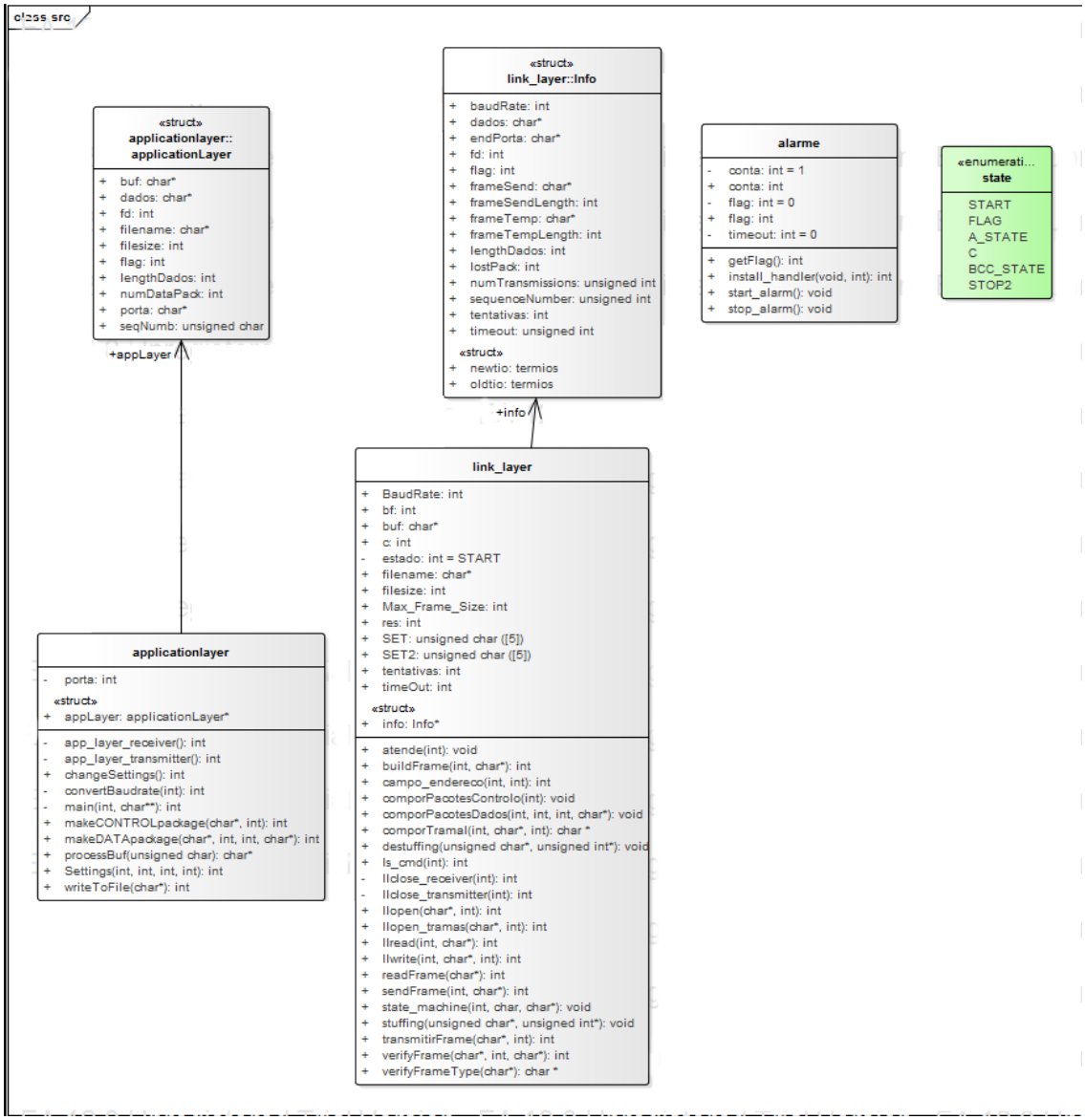



**Diagrama de Classes**

## Anexo II

```
void state_machine(int state, char signal, char * type){


        if (state == START){
                if (signal == F){
                        state = FLAG;
                        SET2[0]=signal;
                }
        }
        else if (state == FLAG){
                if (signal == F)
                        state = FLAG;
                else if ((signal == campo_endereco(!info->flag, C_SET) && type == "set")
                  || (signal == campo_endereco(!info->flag, C_UA) && type == "ua")
                  || (signal == campo_endereco(!info->flag, C_DISC) && type == "disc")
                  || (signal == campo_endereco(!info->flag, RR(1)) && type == "rr1")
                  || (signal == campo_endereco(!info->flag, RR(0)) && type == "rr0")
                  || (signal == campo_endereco(!info->flag, REJ(1)) && type == "rej1")
                  || (signal == campo_endereco(!info->flag, REJ(0)) && type == "rej0")
                  || (signal == campo_endereco(!info->flag, C_I0) && type == "I0")
                  || (signal == campo_endereco(!info->flag, C_I1) && type == "I1")){
                        state = A_STATE;
                        SET2[1]=signal;
                }
                else
                        state = START;
        }
        else if (state == A_STATE){
                if (signal == F){
                        state = FLAG;
                }
                else if ((signal == C_SET && type == "set")
                  || (signal == C_UA && type == "ua")
                  || (signal == C_DISC && type == "disc")
                  || (signal == RR(1) && type == "rr1")
                  || (signal == RR(0) && type == "rr0")
                  || (signal == REJ(1) && type == "rej1")
                  || (signal == REJ(0) && type == "rej0")
                  || (signal == C_I0 && type == "I0")
                  || (signal == C_I1 && type == "I1")){
                        state = C;
                        SET2[2]=signal;
                }
                else
                        state = START;
        }
        else if (state == C){
                if (signal == F)
                        state = FLAG;
                else if (signal == (SET2[1]^SET2[2])){
                        state = BCC_STATE;
                        SET2[3]=signal;
                }
                else
                        state = START;
        }
        else if (state == BCC_STATE){
                if (signal == F){
                        state = STOP2;
                        SET2[4]=signal;
                }
                else
                        state = START;
        }
        estado = state;

        //printf("estado: %d \n", estado);
}
```

**Máquina de estados implementada**

**Anexo III**

**Alarme.c**

```
#include "alarme.h"

int timeout = 0;

int flag=0, conta=1;


int install_handler(void(*handler)(int), int timeOut){
        struct sigaction sa;
        sigaction(SIGALRM, NULL, &sa);

        timeout = timeOut;

   sa.sa_handler = handler;

   if (sigaction(SIGALRM, &sa, NULL) == -1)
        return -1;
}

void start_alarm(){
        alarm(timeout);
        flag = 0;
}

void stop_alarm(){
        alarm(0);
        flag = 0;
}

int getFlag(){
        return flag;
}
```

**Alarme.h**

```
#pragma once

#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <signal.h>

extern int flag;
extern int conta;
int install_handler(void(*handler)(int), int timeout);
void start_alarm();
void stop_alarm();
int getFlag();
```

**applicationlayer.c**

```
/*
* Aplication Layer
*/
#include "applicationlayer.h"

#define MAX_FRAME_SIZE 100

int porta;

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

	if(argc != 2){
		printf("numero de argumentos errado. \n");
		printf("%s (porta(/dev/ttySN)) \n", argv[0]);
		return 0;
	}
	appLayer = malloc(sizeof(struct applicationLayer));
	appLayer->porta = argv[1];
	appLayer->buf = malloc(1000); //escrevemos sempre no mesmo buffer ele é sempre
reescrito

	int alterouOption = 0;
	while(1){
		printf("*****MENU*****\n");
		printf("1) Transmitir\n");
		printf("2) Receber\n");
		printf("3) Settings\n");
		printf("4) Exit\n");

		int option = 0;
		do{
			scanf("%d", &option);
		} while(option < 1 || option > 4);

		if (option == 4)
			return 0;
		else if (option == 3){
			changeSettings();
			alterouOption = 1;
			continue;
		}
		else{
			if (!alterouOption)
				Settings(3, 3, 38400, 43);
			if(option == 1)
				appLayer->flag = TRANSMITTER;
```

```
                else if (option == 2)
                        appLayer->flag = RECEIVER;

                break;
            }
        }

        if (appLayer->flag == TRANSMITTER){
            char fileC[20];
            printf("File: ");
            scanf("%s", fileC);

            appLayer->filename = fileC; //Nome do ficheiro perguntado no menu

            app_layer_transmitter();
        }
        else if (appLayer->flag == RECEIVER)
            app_layer_receiver();

        free(appLayer->buf);
        free(appLayer);

        return 1;
}

int app_layer_transmitter(){
        appLayer->fd=0;
        if((appLayer->fd=open(appLayer->filename,O_RDONLY,0666)) < 0)
            return 0;

        printf("ficheiro aberto: %s \n", appLayer->filename);

        struct stat fileStat;

        if(fstat(appLayer->fd,&fileStat) < 0){
            printf("Erro no FSTAT\n");
            return 0;
        }

        appLayer->filesize = fileStat.st_size;
        if (appLayer->filesize < 0){
            printf("Erro no file size\n");
            return 0;
        }

        appLayer->lengthDados = (Max_Frame_Size - 2 - 8 -4)/2;
        appLayer->numDataPack = (int)(((float)appLayer->filesize)/appLayer-
>lengthDados+.5);
```

```
	int n1 = makeCONTROLpackage(appLayer->buf,1);


	if(n1==0)
		return 0;

	appLayer->seqNumb = 0;
	int i = 0;
	char * dados = malloc(1000);
	printf("numDataPack = %d \n", appLayer->numDataPack);

	/*
	llopen....
	*/

	int llo = llopen(appLayer->porta, TRANSMITTER);

	llwrite(1, appLayer->buf, n1);

	for(i=0; i <= appLayer->numDataPack ; i++){
		int res;

		do{res = read(appLayer->fd, dados, appLayer->lengthDados); }while(res ==
0);

		int datalength = makeDATApackage(appLayer->buf, appLayer->seqNumb,
res, dados);

		/*
			Escrever aqui o código que usa o link_layer para enviar os dados
			llwrite
		*/
		//llwrite(0, appLayer->buf, datalength);

		int llw = llwrite(appLayer->fd, appLayer->buf,datalength);

		printf("Percentagem de dados enviados: %f \n", ((float)i*100)/appLayer-
>numDataPack);

		appLayer->seqNumb++;
	}

	int n2 = makeCONTROLpackage(appLayer->buf,2);
	llwrite(appLayer->fd, appLayer->buf,n2);


	int llc = llclose_transmitter(appLayer->fd);
```

```
}

int app_layer_receiver(){
	int llo = llopen(appLayer->porta, RECEIVER);
	appLayer->dados = malloc(150);
	appLayer->filename = malloc(150);


	llread(RECEIVER, appLayer->buf);
	if(appLayer->buf[0] != 1){
		printf("pacote de controlo inicial com campo de controlo errado: %d\n",
appLayer->buf[0]);
		return 0;
	}
	int fileSize;


	int j = 1;
	int ite = 0;
	int octSize;
	for(ite = 0; ite < 2; ite++){
		if (appLayer->buf[j] == 0){
			octSize = appLayer->buf[j+1];
			//printf("octSize do fileSize: %d \n", octSize);
			memcpy(&appLayer->filesize, appLayer->buf+(j+2), octSize);
			//printf("fileSize: %d \n", appLayer->filesize);
		}
		else if (appLayer->buf[j] == 1){
			octSize = appLayer->buf[j+1];
			memcpy(appLayer->filename, appLayer->buf+(j+2), octSize);
			appLayer->filename[octSize] = 0;
			//printf("received filename %s\n", appLayer->filename);
		}
		j+= 2+octSize;
	}

	appLayer->fd=0;
	appLayer->fd = open(appLayer->filename, O_CREAT | O_TRUNC | O_WRONLY,
0666);
	if(appLayer->fd < 0 ){
		printf("Não abriu corretamente para escrita: %s\n", appLayer->filename);
		return 0;
	}



	appLayer->lengthDados = (Max_Frame_Size - 2 - 8 -4)/2;
```

```
	appLayer->numDataPack = (int)(((float)appLayer->filesize)/appLayer->lengthDados+.5);
	printf("numDataPack do receiver = %d \n", appLayer->numDataPack);

	int x;
	for(x = 0; x <= appLayer->numDataPack; x++){
		int llr = llread(0, appLayer->buf);
		appLayer->dados = processBuf(appLayer->seqNumb);
		if (appLayer->dados == "rip" || appLayer->dados == 0){
			x--;
			continue;
		}
		printf("Percentagem de dados recebidos: %3f \n", ((float)x*100)/appLayer->numDataPack);
		//printf("escrever no ficherio\n\n\n\n");
		while(!writeToFile(appLayer->dados))
			continue;
		appLayer->seqNumb++;
	}
	//printf("acabaram\n");

	llread(0, appLayer->buf);

	if(appLayer->buf[0] != 2){
		printf("pacote de controlo final com campo de controlo errado: %d\n", appLayer->buf[0]);
		return 0;
	}
	else{
		//printf("ultimo pacote lido\n");
	}


	int llc = llclose_receiver(appLayer->fd);

	printf("Número de pacotes perdidos: %d \n", info->lostPack);

	if (llc){
		//printf("llclose_receiver funcionou \n");
		return 1;
	}
	else
		return 0;

}

// Cria control packages que são enviadas no antes e depois da transferência de dados
int makeCONTROLpackage(char* buf,int c){
```

```
	if (c == 1 || c == 2){
		buf[0] = c; // pacote enviado no início (start) e no final (end)
	}
	else{
		return 0;
	}

	//printf("fileSize: %d \n", appLayer->filesize);

	//primeiro é enviado o tamanho e depois o nome
	buf[1] = 0;
	buf[2] = sizeof(appLayer->filesize);
	memcpy(buf + 3, &appLayer->filesize, sizeof(appLayer->filesize));


	buf[3 +sizeof(appLayer->filesize)] = 1;
	buf[4+sizeof(appLayer->filesize)] = strlen(appLayer->filename);
	memcpy(buf + 5 + sizeof(appLayer->filesize), appLayer->filename, strlen(appLayer-
>filename));

	//int i = 0;
	//printf("trama de controlo %d: ", c);
	/*for (i = 0; i < (4+sizeof(appLayer->filesize)+strlen(appLayer->filename)+1); i++){
		printf("buf[%d] = %x", i, buf[i]);
	}
	printf("\n");*/

	return 4+sizeof(appLayer->filesize)+strlen(appLayer->filename)+1;
}

// Cria data package que envia o ficheiro
int makeDATApackage(char* buf,int seqNumb, int lengthDados, char* dados){
	buf[0] = 0;
	buf[1] = seqNumb;
	buf[2] = lengthDados/256;
	buf[3] = lengthDados%256;
	int i;
	for(i=0; i < lengthDados; i++){
		buf[4+i] = dados[i];
	}
	return (4+i);
}

int writeToFile(char* dados){

	int res = write(appLayer->fd, dados,  256 * appLayer->buf[2] + appLayer->buf[3]);
	//printf("Writing to ficherio %d\n", 256 * appLayer->buf[2] + appLayer->buf[3]);
```

```
	//write(STDIN_FILENO, dados,  256 * appLayer->buf[2] + appLayer->buf[3]);
	//printf("\ndone writing to ficherio\n");
	if(res == 0)
		return 0;

	else return 1;
}


char* processBuf(unsigned char seqnumb){

	if(appLayer->buf[0] != 0)
		return 0;

	if(appLayer->buf[1] != (char)seqnumb){
		//printf("rip seqnumb. buf[1] = 0x%x em vez de seqnumb = 0x%x\n",
appLayer->buf[1], seqnumb);
		return "rip";
	}

	char * bf = malloc(150);
	int i=0;
	for(i=0; i< 256 * appLayer->buf[2] + appLayer->buf[3]; i++) {
		bf[i] =  appLayer->buf[4 + i];
	}
	return bf;

}

int changeSettings(){
	char fileC[20];
	int tentativasC;
	int timeOutC;
	int baudRateC;
	int max_size;

	printf("tentativas: ");
	scanf("%d", &tentativasC);

	printf("timeOut: ");
	scanf("%d", &timeOutC);

	printf("BaudRate: ");
	scanf("%d", &baudRateC);

	printf("Max_Frame_Size: ");
	scanf("%d", &max_size);
```

```
        Settings(tentativasC, timeOutC, baudRateC, max_size);
        return 1;

}

int Settings(int trys, int timeO, int BR, int FrameSize){
        tentativas = trys;
        timeOut = timeO;
        BaudRate = convertBaudrate(BR);
        Max_Frame_Size = 2*FrameSize+2+4+8;

        return 1;
}

int convertBaudrate(int baudrate){
        switch(baudrate){
        case 300:
                return B300;
        case 1200:
                return B1200;
        case 2400:
                return B2400;
        case 4800:
                return B4800;
        case 9600:
                return B9600;
        case 19200:
                return B19200;
        case 38400:
                return B38400;
        case 57600:
                return B57600;
        case 115200:
                return B115200;
        case 230400:
                return B230400;
        default:
                return -1;
         }
}
```

**applicationlayer.h**

```
/*
* Application Layer .h
*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include "link_layer.h"

struct applicationLayer {
	int fd; // descritor de ficheiro
	int flag; /*TRANSMITTER | RECEIVER*/

	char* filename; //file name
	int filesize; //file size
	int lengthDados;
	char* buf; //escrevemos sempre no mesmo buffer ele é sempre reescrito
	int numDataPack;
	unsigned char seqNumb;
	char * dados;
	char * porta;
};

struct applicationLayer * appLayer;

int makeCONTROLpackage(char* buf,int c);
int makeDATApackage(char* buf,int seqNumb, int lengthDados, char* dados);
int writeToFile(char* dados);
char* processBuf(unsigned char seqnumb);
int changeSettings();
int Settings(int trys, int timeO, int BR, int FrameSize);
```

**link_layer.c**

```
#include "link_layer.h"

int estado = START;

/*
  Lê uma frame para o parametro "frame" e retorna o tamanho da frame
*/
int readFrame(char * frame){
  char buf2 = 0;
  int res2;
  int i = 0;
  int j = 0;
  int primeiroF = 1; //passa a 0 assim que a primeira FLAG é encontrada
  //printf("Received: ");
  while(1){
    while((res2 = read(info->fd, &buf2, 1))==0)
      continue;

    if (res2 == -1)
      return 0;

    //start_alarm();

    //printf("0x%02x ", buf2);


    if (buf2 == F){
      if (!primeiroF){  //se não for a primeira FLAG, será a ultima e termina
        if (i < 4){ //se a trama não tiver pelo menos 5 chars é invalida e começa a ler outra
          i = 1;
          frame[0] = buf2;
          continue;
        }
        else{ //se a trama tiver tamanho 5 ou superior termina
          frame[i] = buf2;
          i++;
          break;
        }
      }
      else{  //se for a primeira FLAG, começa a ler.
        primeiroF = 0;
        frame[i] = buf2;
        i++;
        continue;
      }
```

```
    }
    else if (!primeiroF){ //se não for FLAG e ja tiver sido encontrada primeira FLAG, adiciona à
frame
      frame[i] = buf2;
    }
    else
      continue;


    i++;
  }

  //printf("\n");
  return i;
}

/*
  retorna o tipo de frame verificando o campo de controlo
*/
char * verifyFrameType(char * frame){
  switch(frame[2]){
    case C_SET:
      return "set";
      break;
    case C_DISC:
      return "disc";
      break;
    case C_UA:
      return "ua";
      break;
    case RR(1):
      return "rr1";
      break;
    case RR(0):
      return "rr0";
      break;
    case REJ(0):
      return "rej0";
      break;
    case REJ(1):
      return "rej1";
      break;
    case C_I0:
      return "I0";
      break;
    case C_I1:
      return "I1";
      break;
```

```
  }
}

/*
  Verifica se a frame está correta segundo o seu tipo ("type")
*/
int verifyFrame(char * frame, int length, char * type){
  int i=0;
  int j=0;
  estado = START;
  char BBC2 = 0;
  for(i = 0; i < length; i++){

   //printf("verifyFrame %s[%d]: %x \n", type, i, frame[i]);

    if (type != "I0" && type != "I1"){
      if (length != 5){
        info->lostPack++;
        printf("frame do tipo %s com tamanho irregular = %d \n", type, length);
        return 0;
      }
      state_machine(estado, frame[i], type);

    }
    else{
      if (i < 4 || (i == (length-1))){
        state_machine(estado, frame[i], type);
      }
      else if (i == (length - 2)){
        if (frame[i] != BBC2){
            info->lostPack++;
          //printf("\n\n\n\nBCC2 devia ser 0x%02x, mas é 0x%02x\n\n\n\n\n", BBC2, frame[i]);
          return 0;
        }
      }
      else{
        BBC2 = BBC2^frame[i];
        info->dados[j] = frame[i];
        j++;
        info->lengthDados = j;
      }
    }

  }

  if (estado == STOP2){
    estado = START;
    return 1;
```

```
  }
  else{
    info->lostPack++;
    estado = START;
    return 0;
  }
}

/*
  Cria e envia uma trama do tipo type (Não funcional para tipo I)
*/
int buildFrame(int flag, char * type){
  info->frameSend[0] = F;
  if (type == "set")
    info->frameSend[2] = C_SET;
  else if (type == "ua")
    info->frameSend[2] = C_UA;
  else if (type == "disc")
    info->frameSend[2] = C_DISC;
  else if (!strcmp(type, "rr1"))
    info->frameSend[2] = RR(1);
  else if (!strcmp(type,"rr0"))
    info->frameSend[2] = RR(0);
  else if (!strcmp(type,"rej0"))
    info->frameSend[2] = REJ(0);
  else if (!strcmp(type,"rej1"))
    info->frameSend[2] = REJ(0);
  else
    return 0;
  info->frameSend[1] = campo_endereco(info->flag, info->frameSend[2]);
  info->frameSend[3] = info->frameSend[1]^info->frameSend[2];
  info->frameSend[4] = F;
  info->frameSendLength = 5;

  //printf("Trama composta %s: 0x%x 0x%x 0x%x 0x%x 0x%x \n", type, info->frameSend[0],
info->frameSend[1], info->frameSend[2], info->frameSend[3], info->frameSend[4]);

  return 1;
}

char * comporTramaI(int flag, char * buffer, int length){
  /*
    int ecx = 0;

    printf("tamanho dos dados a enviar: %d \n Dados: ", length);
    for(ecx = 0; ecx < length; ecx++){
      printf("%x.", buffer[ecx]);
    }
```

```
    printf("\n");
  */

  int index;
  info->frameSend[0] = F;
  if (info->sequenceNumber == 1){
    info->frameSend[2] = C_I1;
  }
  else
    info->frameSend[2] = C_I0;
  info->frameSend[1] = campo_endereco(flag, info->frameSend[2]);
  info->frameSend[3] = info->frameSend[1]^info->frameSend[2];
  info->frameSend[4 + length] = 0;
  for(index = 0; index < length; index++){
    info->frameSend[4 + index] = buffer[index];
    info->frameSend[4 + length] = info->frameSend[4 + length]^info->frameSend[4 + index];
  }
  info->frameSend[4 + length + 1] = F;
  /*
    int i;
    for(i = 0; i <= (5+length); i++){
      printf("I[%d]=%x ", i, info->frameSend[i]);
    }
    printf("\n");
  */
  info->frameSendLength = 6+length;
  //fprintf(stderr,"Construida ");
 /* int i;
  for(i = 0; i < info->frameSendLength; i++){
    fprintf(stderr,"0x%x ", info->frameSend[i]);
  }
  fprintf(stderr,"\n");*/
  return info->frameSend;
}

int llopen(char * porta, int flag){

  printf("FLAG (TRANS/REC) = %d \n", flag);
  info = malloc(sizeof(struct Info));
  info->dados = malloc(255);
  info->frameTemp = malloc(255);
  info->frameSend = malloc(255);
  info->timeout = timeOut;
  install_handler(atende, info->timeout);
  //printf("sequenceNumber: %d \n", info->sequenceNumber);
  info->tentativas = tentativas;
  info->flag = flag;
```

```
info->endPorta = malloc(255);
info->endPorta = porta;
info->fd = open(info->endPorta, O_RDWR | O_NOCTTY);
if (info->fd < 0) {perror(info->endPorta); exit(-1);}

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

bzero(&info->newtio, sizeof(info->newtio));

info->newtio.c_cflag = BaudRate | CS8 | CLOCAL | CREAD;
info->newtio.c_iflag = IGNPAR;
info->newtio.c_oflag = OPOST;

// set input mode (non-canonical, no echo,...)
info->newtio.c_lflag = 0;

info->newtio.c_cc[VTIME]    = 0;   // inter-character timer unused
info->newtio.c_cc[VMIN]     = 1;   // blocking read until 5 chars received

tcflush(info->fd, TCIFLUSH);

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

if (flag == TRANSMITTER){
  printf("llopen de transmissor \n");
  buildFrame(flag, "set");
  transmitirFrame(info->frameSend, info->frameSendLength);
  while(info->tentativas > 0){
    //printf("tentativasOpen = %d \n", info->tentativas);
    start_alarm();
    info->frameTempLength = readFrame(info->frameTemp);
    if (verifyFrame(info->frameTemp, info->frameTempLength, "ua")){
      stop_alarm();
      info->tentativas = tentativas;
      return 1;
    }
  }
  if (info->tentativas == 0){
        printf("Número de tentativas chegou ao fim. \n");
        exit(-1);
  }
}
```

```
  else{
    printf("llopen de recetor \n");
    info->frameTempLength = readFrame(info->frameTemp);
    if (verifyFrame(info->frameTemp, info->frameTempLength, "set")){
      buildFrame(flag, "ua");
      transmitirFrame(info->frameSend, info->frameSendLength);
      //printf("terminar llopen recetor \n");
      return 1;
    }
  }

  return info->fd;
}

int llwrite(int fd, char * buffer, int length){

  comporTramaI(TRANSMITTER, buffer, length);
  stuffing(info->frameSend, &info->frameSendLength);
  //printf("partes: %x, %x, %x, %x, %x, %x, %x, %x, %x \n",
tramaI[0],tramaI[1],tramaI[2],tramaI[3],tramaI[4],tramaI[5],tramaI[6],tramaI[7],tramaI[8]);
  transmitirFrame(info->frameSend, info->frameSendLength);
  //printf("enviar frame I com sequenceNumber = %d \n", info->sequenceNumber);
  info->tentativas = tentativas;
  while(info->tentativas > 0){
    start_alarm();
    info->frameTempLength = readFrame(info->frameTemp);
    if (info->sequenceNumber == 1){
      if (verifyFrame(info->frameTemp, info->frameTempLength, "rr0")){
        //printf("recebeu rr corretamente \n");
        stop_alarm();
        info->tentativas = tentativas;
        break;
      }
      else if (verifyFrame(info->frameTemp, info->frameTempLength, "rej0")){
        //printf("recebeu rej0\n");
        transmitirFrame(info->frameSend, info->frameSendLength);
        continue;
      }
    }
    else if (info->sequenceNumber == 0){
      if (verifyFrame(info->frameTemp, info->frameTempLength, "rr1")){
        //printf("recebeu rr corretamente \n");
        stop_alarm();
        info->tentativas = tentativas;
        break;
      }
      else if (verifyFrame(info->frameTemp, info->frameTempLength, "rej1")){
        //printf("recebeu rej1\n");
```

```
        transmitirFrame(info->frameSend, info->frameSendLength);
        continue;
      }
    }
  }
  if (info->tentativas == 0){
          printf("Número de tentativas chegou ao fim. \n");
          exit(-1);
  }
  info->sequenceNumber = !info->sequenceNumber;
  //printf("retornar llwrite\n");
  return 1;
}

int llread(int fd, char * buffer){

  //printf("iniciar llread \n");
  while(1){
    info->frameTempLength = readFrame(info->frameTemp);
    char * type = NULL;
    type = verifyFrameType(info->frameTemp);
    if (type == "set"){
      buildFrame(info->flag, "ua");
      transmitirFrame(info->frameSend, info->frameSendLength);
      continue;
    }
    else if (type == "I0" || type == "I1"){
      destuffing(info->frameTemp, &info->frameTempLength);
        //fprintf(stderr,"Destuffing ");
        /*int bb;
        for(bb = 0; bb < info->frameTempLength; bb++){
          fprintf(stderr,"0x%x ", info->frameTemp[bb]);
        }
        fprintf(stderr,"\n");*/
      if (verifyFrame(info->frameTemp, info->frameTempLength, type)){

        if(type == "I0" && !info->sequenceNumber
          || type == "I1" && info->sequenceNumber){
          //printf("recebeu a trama I correspondente aos sequenceNumber %d \n", info-
>sequenceNumber);
          char * typeRR = malloc(5);
          sprintf(typeRR, "rr%d", !info->sequenceNumber);
          //printf("criar frame de %s \n", typeRR);
          buildFrame(info->flag, typeRR);
          transmitirFrame(info->frameSend, info->frameSendLength);
          free(typeRR);
          info->sequenceNumber = !info->sequenceNumber;
        }
```

```
        else{
          char * typeRR = malloc(5);
          sprintf(typeRR, "rr%d", info->sequenceNumber);
          //printf("criar frame de %s \n", typeRR);
          buildFrame(info->flag, typeRR);
          transmitirFrame(info->frameSend, info->frameSendLength);
          free(typeRR);
          continue;
        }

        int j;
        //printf("frameTempLength: %d\n", info->frameTempLength);
        //printf("dados recebidos: ");


        for(j = 0; j < (info->frameTempLength-6); j++){
          info->dados[j] = info->frameTemp[4+j];
          //printf(" %x ", info->dados[j]);
          buffer[j] = info->dados[j];
          //printf(" %x \n", info->dados[j]);
        }
        //printf("\n");
        info->lengthDados = j;
      }
      else{
        char * typeREJ = malloc(5);
        sprintf(typeREJ, "rej%d", !info->sequenceNumber);
        //printf("criar frame de %s \n", typeREJ);
        buildFrame(info->flag, typeREJ);
        transmitirFrame(info->frameSend, info->frameSendLength);
        free(typeREJ);
           continue;
      }
    }
    break;
  }

  return 1;
}

int llclose_transmitter(int fd){
  info->tentativas = tentativas;

  while(info->tentativas > 0){
    buildFrame(info->flag, "disc");
    transmitirFrame(info->frameSend, info->frameSendLength);
    start_alarm();
```

```
    info->frameTempLength = readFrame(info->frameTemp);
    char * type = malloc(5);
    type = verifyFrameType(info->frameTemp);
    if (verifyFrame(info->frameTemp, info->frameTempLength, "disc")){
      buildFrame(info->flag, "ua");
      if(transmitirFrame(info->frameSend, info->frameSendLength))
        break;
    }
  }

  if (info->tentativas == 0){
          printf("Número de tentativas chegou ao fim. \n");
          exit(-1);
  }

   sleep(1);
    if ( tcsetattr(info->fd,TCSANOW,&info->oldtio) == -1) {
      perror("tcsetattr");
      return -1;
    }
    close(fd);
    printf("fechou transmissor\n");
    return 1;
}

int llclose_receiver(int fd){
  info->tentativas = tentativas;
  while(1){
    info->frameTempLength = readFrame(info->frameTemp);
    char * type = malloc(5);
    type = verifyFrameType(info->frameTemp);

    if (type == "I0" || type == "I1"){
      if (verifyFrame(info->frameTemp, info->frameTempLength, type)){
        char * typeRR = malloc(5);
        sprintf(typeRR, "rr%d", !info->sequenceNumber);
        //printf("criar frame de %s \n", typeRR);
        buildFrame(info->flag, typeRR);
        transmitirFrame(info->frameSend, info->frameSendLength);
        free(typeRR);
        int j;
        //printf("dados recebidos: ");
        for(j = 0; j < (info->frameTempLength-6); j++){
          info->dados[j] = info->frameTemp[4+j];
          //printf(" %x ", info->dados[j]);
        }
        //printf("\n");
        info->lengthDados = j;
```

```
        continue;
      }
      else{
        char * typeREJ = malloc(5);
        sprintf(typeREJ, "rej%d", !info->sequenceNumber);
        //printf("criar frame de %s \n", typeREJ);
        buildFrame(info->flag, typeREJ);
        transmitirFrame(info->frameSend, info->frameSendLength);
        free(typeREJ);
        continue;
      }
    }
    else if (verifyFrame(info->frameTemp, info->frameTempLength, "disc")){
      buildFrame(info->flag, "disc");
      transmitirFrame(info->frameSend, info->frameSendLength);
      start_alarm();

      info->frameTempLength = readFrame(info->frameTemp);
      type = verifyFrameType(info->frameTemp);

      if (verifyFrame(info->frameTemp, info->frameTempLength, "ua")){
        break;
      }
    }
    else{
      printf("llclose_receiver não recebeu nem I nem disc \n");
    }
  }


  if ( tcsetattr(info->fd,TCSANOW,&info->oldtio) == -1) {
    perror("tcsetattr");
    return 0;
  }
  close(fd);
  printf("fechou recetor\n");
  return 1;
}

int transmitirFrame(char * frame, int length){
  int i;
  //fprintf(stderr, "Enviar frame tamanho %d : ", length);
  for(i = 0; i < length; i++){
    res = write(info->fd,&frame[i],1);
    if (res == 0 || res == -1)
      return 0;
    //fprintf(stderr,"0x%x ", frame[i]);
  }
```

```
  //fprintf(stderr,"\n");
  return 1;
}

void state_machine(int state, char signal, char * type){


	if (state == START){
		if (signal == F){
			state = FLAG;
			SET2[0]=signal;
		}
	}
	else if (state == FLAG){
		if (signal == F)
			state = FLAG;
		else if ((signal == campo_endereco(!info->flag, C_SET) && type == "set")
		  || (signal == campo_endereco(!info->flag, C_UA) && type == "ua")
		  || (signal == campo_endereco(!info->flag, C_DISC) && type == "disc")
		  || (signal == campo_endereco(!info->flag, RR(1)) && type == "rr1")
		  || (signal == campo_endereco(!info->flag, RR(0)) && type == "rr0")
		  || (signal == campo_endereco(!info->flag, REJ(1)) && type == "rej1")
		  || (signal == campo_endereco(!info->flag, REJ(0)) && type == "rej0")
		  || (signal == campo_endereco(!info->flag, C_I0) && type == "I0")
		  || (signal == campo_endereco(!info->flag, C_I1) && type == "I1")){
			state = A_STATE;
			SET2[1]=signal;
		}
		else
			state = START;
	}
	else if (state == A_STATE){
		if (signal == F){
			state = FLAG;
		}
		else if ((signal == C_SET && type == "set")
		  || (signal == C_UA && type == "ua")
		  || (signal == C_DISC && type == "disc")
		  || (signal == RR(1) && type == "rr1")
		  || (signal == RR(0) && type == "rr0")
		  || (signal == REJ(1) && type == "rej1")
		  || (signal == REJ(0) && type == "rej0")
		  || (signal == C_I0 && type == "I0")
		  || (signal == C_I1 && type == "I1")){
			state = C;
			SET2[2]=signal;
		}
		else
```

```
                state = START;
        }
        else if (state == C){
                if (signal == F)
                        state = FLAG;
                else if (signal == (SET2[1]^SET2[2])){
                        state = BCC_STATE;
                        SET2[3]=signal;
                }
                else
                        state = START;
        }
        else if (state == BCC_STATE){
                if (signal == F){
                        state = STOP2;
                        SET2[4]=signal;
                }
                else
                        state = START;
        }
        estado = state;

        //printf("estado: %d \n", estado);
}

int campo_endereco(int role, int c){
  if (role == TRANSMITTER){
    if (Is_cmd(c)){
      return 0x03;
    }
    else{
      return 0x01;
    }
  }
  else if (role == RECEIVER){
    if (Is_cmd(c)){
      return 0x01;
    }
    else{
      return 0x03;
    }
  }

  //printf("fail no campo_endereco \n");
}

int Is_cmd(int comand){
  if (comand == C_I0 || comand == C_I1 || comand == C_SET || comand == C_DISC)
```

```
    return 1;
  else
    return 0;
}

void comporPacotesControlo(int c){

 buf[0] = c;
  //primeiro vou por o tamanho e depois o nome
 buf[1] = 0;
 buf[2] = sizeof(filesize);
 memcpy(buf+3, &filesize, sizeof(filesize));

 buf[3 +sizeof(filesize)] = 1;
 buf[5] = sizeof(filename);
 memcpy(buf + 4 +sizeof(filesize), &filesize, strlen(filename));
}

void comporPacotesDados(int seqNumb, int sizeCampoI, int lengthDados, char* dados){

 buf[0] = 0;
 buf[1] = seqNumb;
 buf[2] = lengthDados/256;
 buf[3] = lengthDados%256;
  int i;
  for(i=0; i < lengthDados; i++){

 buf[4] = dados[i];

  }
}

void atende(int sig) {
  printf("alarme # %d\n", conta);
  flag=1;
  conta++;
  printf("tentativas = %d \n", info->tentativas);
  if (info->tentativas > 0){
    transmitirFrame(info->frameSend, info->frameSendLength);
    info->tentativas--;
  }
  else{
    fprintf(stderr, "0 tentativas restantes \n");
    stop_alarm();
    exit(-1);
  }
}
```

```
// Stuffing
void stuffing(unsigned char* frame, unsigned int* size){
  int i;
  for (i = 1; i < (*size-1); i++){
      if (frame[i] == 0x7e){

        memmove(frame + i + 1,frame + i ,*size-i);
        frame[i] = 0x7d;
        frame[++i] = 0x5e;
        (*size)++;
      }
      else if (frame[i] == 0x7d){

        memmove(frame + i + 1,frame + i ,*size-i);
        frame[i] = 0x7d;
        frame[++i] = 0x5d;
        (*size)++;
      }
  }
}

//DESTUFFING
void destuffing(unsigned char* frame, unsigned int* size){
  int i;
  for (i = 1; i < (*size-1); i++){
    if(frame[i] == 0x7d && frame[i+1] == 0x5e){

       memmove(frame + i + 1, frame + i + 2, *size-i-2);
       frame[i] = 0x7e;
      (*size)--;
    }
    else if (frame[i] == 0x7d && frame[i+1]== 0x5d){

      memmove(frame + i + 1, frame + i + 2, *size-i-2);
      //frame[i] = 0x7d;
     (*size)--;
    }

  }
}
```

**link_layer.h**

```
#pragma once

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include "alarme.h"

#define BAUDRATE B38400
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define F 0x7E
#define A 0x03
#define C_SET 0x07
#define BCC (A^C_SET)
#define C_UA 0x03
#define C_DISC 0x0B
#define C_I0 0x0
#define C_I1 0x20
#define TRANSMITTER 1
#define RECEIVER 0
#define RR(N) (N<<5 | 1)
#define REJ(N) (N<<5 | 5)

struct Info {
  int fd; // descritor de ficheiro
  struct termios oldtio;
  struct termios newtio;
  char * endPorta; /*Dispositivo /dev/ttySx, x = 0, 1*/

  int baudRate; /*Velocidade de transmissão*/
  unsigned int sequenceNumber; /*Número de sequência da trama: 0, 1*/
  unsigned int timeout; /*Valor do temporizador: 1 s*/
  unsigned int numTransmissions; /*Número de tentativas em caso de falha*/

  int flag;

  char * dados; /*dados a enviar/receber*/
  int lengthDados;
  int tentativas;
```

```
  char * frameTemp; // serve para guardar uma frame temporariamente
  int frameTempLength;
  char * frameSend;
  int frameSendLength;

  int lostPack;

};

//volatile int STOP=FALSE;
int tentativas;
int timeOut;
int Max_Frame_Size;
int BaudRate;
unsigned char SET[5];
unsigned char SET2[5];

struct Info * info;
int c, res;
char* buf; //file buffer
int bf;
char* filename; //file name
int filesize; //file size

//STATES
enum state {START, FLAG, A_STATE, C, BCC_STATE, STOP2};
//int estado = START;

int readFrame(char * frame);
char * verifyFrameType(char * frame);
int verifyFrame(char * frame, int length, char * type);
int llopen(char * porta, int flag);
void state_machine(int state, char signal, char * type);
int llopen_tramas(char * frame, int flag);
int sendFrame(int flag, char * type);
int campo_endereco(int role, int c);
int Is_cmd(int comand);
void comporPacotesControlo(int c);
void comporPacotesDados(int seqNumb, int sizeCampoI, int lengthDados, char* dados);
int transmitirFrame(char * frame, int length);
void atende(int sig);
int llwrite(int fd, char * buffer, int length);
int llread(int fd, char * buffer);
char * comporTramaI(int flag, char * buffer, int length);
int buildFrame(int flag, char * type);
void stuffing(unsigned char* frame, unsigned int* size);
void destuffing(unsigned char* frame, unsigned int* size);
```

**controlpackages.c**

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

int fd, control;

// SEND THE CONTROL PACKAGE
void send_CONTROL_pck(int arg, int tamanho, char nome){
   if (arg == 1){ // start package
        int t = 0;
        control = write(fd, arg, 1); // enviar o C

        /* ENVIO DO TAMANHO DO FICHEIRO (T1) */
        control = write(fd, t, 1); // enviar o T1
        control = write(fd, sizeof(tamanho), 1); // enviar o tamanho em octetos do valor
        unsigned int i = 0;
        while(i < sizeof(tamanho)){ // enviar o V
            control = write(fd, tamanho, sizeof(tamanho));
            ++i;
        }

        /* ENVIO DO NOME DO FICHEIRO (T2) */
        control = write(fd, t++, 1); // enviar o T2
        control = write(fd, sizeof(nome), 1); // enviar o tamanho do nome do ficheiro
        int k = 0;
        while(k < sizeof(nome)){
                control = write(fd, &nome, sizeof(nome)); // enviar o nome do ficheiro
                ++k;
        }

   }
   else if (arg == 2){ // end package

   }
   else{ // error case
      printf("control package argument invalid");
   }
}

void send_DATA(){
```

```
}

/* STUFFING */
void stuffing(unsigned char* frame, unsigned int* size){
        for (int i = 1; i < (*size-1); i++){
            if (frame[i] == 0x7e){
                frame[i] = 0x7d;
                memcpy(frame + i+2,frame+i+1,*size-i-1);
                frame[i++] = 0x5e;
                (*size)++;
            }
            else if (frame[i] == 0x7d){
                frame[i] = 0x7d;
                memcpy(frame + i + 2,frame + i + 1,*size-i-1);
                frame[i++] = 0x5d;
                (*size)++;
                }
        }
}

/* DESTUFFING */
void destuffing(unsigned char* frame, unsigned int* size){
        for(int i = 1; i < (*size-1); ++i){
                if(frame[i] == 0x7d && frame[i++] == 0x5e){
                  frame[i] = 0x7e;
                  memcpy(frame + i + 1, frame + i + 2, *size-i-1);
                  (*size--);
                }
                else if (frame[i] == 0x7d && frame[i++]== 0x5d){
                        frame[i] = 0x7d;
                        memcpy(frame + i + 1, frame + i + 2, *size-i-1);
                        (*size--);
                }
        }
}

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


   //send_CONTROL_pck(1, 1, 'A');

}
```

**llfunctions.c**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <signal.h>
#include "alarme.h"

#define BAUDRATE B38400
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define F 0x7E
#define A 0x03
#define C_SET 0x07
#define BCC (A^C_SET)
#define C_UA 0x03
#define C_DISC 0x0B
#define C_I0 0x0
#define C_I1 0x20
#define TRANSMITTER 1
#define RECEIVER 0

struct Info {
  int fd; // descritor de ficheiro
  struct termios oldtio;
  struct termios newtio;
  char endPorta[20]; /*Dispositivo /dev/ttySx, x = 0, 1*/

  int baudRate; /*Velocidade de transmissão*/
  unsigned int sequenceNumber; /*Número de sequência da trama: 0, 1*/
  unsigned int timeout; /*Valor do temporizador: 1 s*/
  unsigned int numTransmissions; /*Número de tentativas em caso de falha*/

  char * dados; /*dados a enviar/receber*/
  int lengthDados;

};


int llopen(int porta, int flag);
int llclose(int fd);
int llclose_transmitter(int fd);
```

```
int llclose_receiver(int fd);
void state_machine(int state, char signal, char * type);
int trasmitirSET(int flag, char * type);
int receberSET(int flag, char * type);
int llwrite(int fd, char * buffer, int length);
int llread(int fd, char * buffer);
int Is_cmd(int comand);
int campo_endereco(int role, int c);
int transmitirFrame(char * frame, int length);
void stuffing(unsigned char* frame, unsigned int* size);
void destuffing(unsigned char* frame, unsigned int* size);
char * comporTramaI(int flag, char * buffer, int length);
char * receberI(int flag);
void comporPacotesDados(int seqNumb, int sizeCampoI, int lengthDados, char* dados);
void comporPacotesControlo(int c);

volatile int STOP=FALSE;

unsigned char SET[5];
unsigned char SET2[5];

struct Info * info;
int c, res;
char* buf; //file buffer
int bf;
char* filename; //file name
int filesize; //file size

//STATES
enum state {START, FLAG, A_STATE, C, BCC_STATE, STOP2};
int estado = START;

int main(int argc, char** argv){
  info = malloc(sizeof(struct Info));
  info->sequenceNumber = 0;
  info->dados = malloc(255);
  printf("sequenceNumber: %d \n", info->sequenceNumber);
  if (strcmp("0", argv[1])==0){
    llopen(atoi(argv[2]), RECEIVER);
    char * result;
    llread(info->fd, result);
    printf("Result: %s /n", result);
    printf("INICIAR LLCLOSE\n");
    llclose_receiver(info->fd);
  }
  else if (strcmp("1", argv[1])==0){
    llopen(atoi(argv[2]), TRANSMITTER);
    printf("cenas\n");
```

```
    info->dados[0] = 0x11;
    printf("cenas\n");
    info->dados[1] = 0x22;
    info->dados[2] = 0x05;
    sleep(1);
    printf("llwrite de %x, %x, %x \n", info->dados[0], info->dados[1], info->dados[2]);
    info->lengthDados = 3;
    llwrite(info->fd, info->dados, info->lengthDados);
    printf("INICIAR LLCLOSE\n");
    llclose_transmitter(info->fd);
  }

 filename = argv[2];
 int file=0;
  if((file=open(filename,O_RDONLY)) < -1)
       return 1;

 struct stat fileStat;
  if(fstat(file,&fileStat) < 0)
        return 1;

 filesize = fileStat.st_size;

 if(read(file, buf, filesize) < 0)
           return 1;




}

int llopen(int porta, int flag){

  sprintf(info->endPorta, "/dev/ttyS%d", porta);
  info->fd = open(info->endPorta, O_RDWR | O_NOCTTY);
  if (info->fd < 0) {perror(info->endPorta); exit(-1);}

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

  bzero(&info->newtio, sizeof(info->newtio));

  info->newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
  info->newtio.c_iflag = IGNPAR;
  info->newtio.c_oflag = OPOST;
```

```
  // set input mode (non-canonical, no echo,...)
  info->newtio.c_lflag = 0;

  info->newtio.c_cc[VTIME]    = 0;   // inter-character timer unused
  info->newtio.c_cc[VMIN]     = 1;   // blocking read until 5 chars received

  tcflush(info->fd, TCIFLUSH);

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

  int tentativas = 3;

  if (flag == RECEIVER){
    if(receberSET(flag, "set")==1)
      transmitirSET(flag, "ua");
    else
      return -1;

    //llclose_receiver(info->fd);
  }
  else{
    while(tentativas > 0){
      transmitirSET(flag, "set");
      alarm(3);
      if (receberSET(flag, "ua") != 1)
        tentativas--;
      else{
        alarm(0);
        break;
      }
    }
    //llclose_transmitter(info->fd);
  }

  return info->fd;
}

int llclose_transmitter(int fd){

    transmitirSET(1, "disc");
    if (receberSET(1, "disc") == 1)
      transmitirSET(1, "ua");
    else
      return -1;
```

```
    sleep(5);

    if ( tcsetattr(info->fd,TCSANOW,&info->oldtio) == -1) {
      perror("tcsetattr");
      return -1;
    }
    close(fd);
    printf("fechou transmissor\n");
    return 1;
}

int llclose_receiver(int fd){

    if (receberSET(1, "disc") == 1){
      transmitirSET(1, "disc");
      if (receberSET(1, "ua") != 1){
        return -1;
      }
    }
    else
      return -1;

    sleep(5);

    if ( tcsetattr(info->fd,TCSANOW,&info->oldtio) == -1) {
      perror("tcsetattr");
      return -1;
    }
    close(fd);
    printf("fechou recetor\n");
    return 1;
}


int transmitirSET(int flag, char * type){
  SET[0] = F;
  SET[1] = A;
  if (type == "set")
    SET[2] = C_SET;
  else if (type == "ua")
    SET[2] = C_UA;
  else if (type == "disc")
    SET[2] = C_DISC;
  SET[3] = SET[1]^SET[2];
  SET[4] = F;

  int i = 0;
```

```
  while(i < 5){
    res = write(info->fd,&SET[i],1);
    i++;
  }
  i=0;
  printf("Send %s: 0x%x 0x%x 0x%x 0x%x 0x%x \n", type, SET[0], SET[1], SET[2], SET[3],
SET[4]);

  return 0;
}

int receberSET(int flag, char * type){
  char buf2 = 0;
  int res2;
  estado = START;
  int i = 0;
  while(i < 5){
    if (i == 0){
      while((res2 = read(info->fd, &buf2, 1))==0 && buf2!=F)
               continue;
    }
    else
              while((res2 = read(info->fd, &buf2, 1))==0)
       continue;

    printf("Received: %x !!! %d \n", buf2, res2);
    i++;
    //printf("i = %d\n", i);

    state_machine(estado, buf2, type);
    if(estado == STOP2){
      estado = START;
      return 1;
    }

  }
  estado = START;
  return 0;
}

char * receberI(int flag){
  //char * dados;
  //dados = malloc(sizeof(255));
  char buf2 = 0;
  int res2;
  int i;
  estado = START;
  for (i = 0; i < 4; i++){
```

```
    if (i == 0){
      while((res2 = read(info->fd, &buf2, 1))==0 && buf2!=F)
       continue;
    }
    else{
      while((res2 = read(info->fd, &buf2, 1))==0)
       continue;
   }

     printf("ReceivedI[%d]: %x !!! %d \n", i, buf2, res2);
     state_machine(estado, buf2, "I");
  }
  if (estado != BCC_STATE)
    return "fail";

  printf("nao falhou na receçao dos primeiros do I\n");
  char BBC2 = 0;
  i = 0;
  buf2 = 1;
  while(BBC2 != buf2){
    while((res2 = read(info->fd, &buf2, 1))==0)
      continue;
    printf("ReceivedDados[%d]: %x !!! %d \n", i, buf2, res2);
    printf("BBC2=%x -- buf2=%x \n", BBC2, buf2);
    if (BBC2 == buf2)
      break;
    BBC2 = BBC2^buf2;
    info->dados[i] = buf2;
    if (i == 0)
      buf2 = 1;
    i++;
  }

  printf("acabaram os dados\n");
  while((res2 = read(info->fd, &buf2, 1))==0)
    continue;
  //state_machine(estado, buf2, "I");
  if (estado == STOP2){
    printf("recebeu a trama I corretamente\n");
    return info->dados;
  }
  else
    return "fail";

}

void state_machine(int state, char signal, char * type){
```

```
if (state == START){
        if (signal == F){
                state = FLAG;
                SET2[0]=signal;
        }
}
else if (state == FLAG){
        if (signal == F)
                state = FLAG;
        else if ((signal == A && type != "I")
          || (signal == campo_endereco(flag, info->sequenceNumber) && type == "I")){
                state = A_STATE;
                SET2[1]=signal;
        }
        else
                state = START;
}
else if (state == A_STATE){
        if (signal == F){
                state = FLAG;
        }
        else if ((signal == C_SET && type == "set")
          || (signal == C_UA && type == "ua")
          || (signal == C_DISC && type == "disc")
          || (signal == RR(0) && type == "rr1")
          || (signal == RR(1) && type == "rr0")
          || (signal == info->sequenceNumber && type == "I")){
                state = C;
                SET2[2]=signal;
        }
        else
                state = START;
}
else if (state == C){
        if (signal == F)
                state = FLAG;
        else if (signal == (SET2[1]^SET2[2])){
                state = BCC_STATE;
                SET2[3]=signal;
        }
        else
                state = START;
}
else if (state == BCC_STATE){
        if (signal == F){
                state = STOP2;
                SET2[4]=signal;
```

```
            }
            else
                state = START;
        }
        estado = state;

        printf("estado: %d \n", estado);
}

int llwrite(int fd, char * buffer, int length){
  char * tramaI;
  //strcpy(tramaI, comporTramaI(TRANSMITTER, buffer, length));
  tramaI = comporTramaI(TRANSMITTER, buffer, info->lengthDados);
  printf("partes: %x, %x, %x, %x, %x, %x, %x, %x, %x \n",
tramaI[0],tramaI[1],tramaI[2],tramaI[3],tramaI[4],tramaI[5],tramaI[6],tramaI[7],tramaI[8]);
  transmitirFrame(tramaI, 6+length);
  alarm(3);
  free(tramaI);
  if (info->sequenceNumber == 1){
    if (receberSET(TRANSMITTER, "rr1")){
      printf("recebeu rr corretamente \n");
      alarm(0);
    }
  }
  else if (info->sequenceNumber == 0){
    if (receberSET(TRANSMITTER, "rr0")){
      printf("recebeu rr corretamente \n");
      alarm(0);
    }
  }
  printf("retornar llwrite\n");
  return 1;
}

int llread(int fd, char * buffer){
  //char * dados;
  //dados = receberI(RECEIVER);
  receberI(RECEIVER);
  printf("Dados recebidos: %x, %x, %x \n", info->dados[0],info->dados[1],info->dados[2]);
  if (info->dados == "fail"){
    //enviar frame REJ
    fprintf(stderr, "falhou a receber a I: %s \n", info->dados);
    return 0;
  }
  char * rrtype = malloc(5);
  printf("cenas dos rr\n");
  sprintf(rrtype, "rr%d", info->sequenceNumber+1);
  fprintf(stderr, "enviar %s\n", rrtype);
```

```
  transmitirSET(RECEIVER, rrtype);
  free(rrtype);
  return 1;
}

char * comporTramaI(int flag, char * buffer, int length){
  char * trama;
  trama = malloc(sizeof(5 + length));
  int index;
  trama[0] = F;
  trama[1] = campo_endereco(flag, info->sequenceNumber);
  trama[2] = info->sequenceNumber;
  trama[3] = trama[1]^trama[2];
  trama[4 + length] = 0;
  for(index = 0; index < length; index++){
    /*
      ADICIONAR STUFFING E DESTUFFING
    */
    trama[4 + index] = buffer[index];
    trama[4 + length] = trama[4 + length]^trama[4 + index];
  }
  trama[4 + length + 1] = F;

  int i;
  for(i = 0; i <= (5+length); i++){
    printf("I[%d]=%x ", i, trama[i]);
  }
  printf("\n");
  return trama;
}

/*
//DESTUFFING feito pela Filipa
void destuffing(unsigned char* frame, unsigned int* size){
    if(frame[i] == 0x7d && frame[i++] == 0x5e){
       frame[i] = 0x7e;
       memcpy(frame + i + 1, frame + i + 2, *size-i-1);
      (*size--);
    }
    else if (frame[i] == 0x7d && frame[i++]== 0x5d){
      frame[i] = 0x7d;
      memcpy(frame + i + 1, frame + i + 2, *size-i-1);
     (*size--);
    }
  }
}

//STUFFING feito pela Filipa
```

```
void stuffing(unsigned char* frame, unsigned int* size){
  for (int i = 1; i < (*size-1); i++){
      if (frame[i] == 0x7e){
    frame[i] = 0x7d;
    memcpy(frame + i+2,frame+i+1,*size-i-1);
    frame[i++] = 0x5e;
    (*size)++;
      }
      else if (frame[i] == 0x7d){
    frame[i] = 0x7d;
    memcpy(frame + i + 2,frame + i + 1,*size-i-1);
    frame[i++] = 0x5d;
    (*size)++;
    }
  }
 }

*/
int transmitirFrame(char * frame, int length){
  int i;
  fprintf(stderr, "Enviar frame tamanho %d : ", length);
  for(i = 0; i < length; i++){
    res = write(info->fd,&frame[i],1);
    fprintf(stderr,"0x%x ", frame[i]);
  }
  fprintf(stderr,"/n");
}

int RR(int N){
  return (N<<5 | 1);
}

int REJ(int N){
  return (N<<5 | 5);
}

int campo_endereco(int role, int c){
  if (role == TRANSMITTER){
    if (Is_cmd(c))
      return 0x03;
    else
      return 0x01;
  }
  else if (role == RECEIVER){
    if (Is_cmd(c))
      return 0x01;
    else
      return 0x03;
```

```
  }

  printf("fail no campo_endereco \n");
}

int Is_cmd(int comand){
  if (comand == C_I0 || comand == C_I1 || comand == C_SET || comand == C_DISC)
    return 1;
  else
    return 0;

}



void comporPacotesControlo(int c){

 buf[0] = c;
//primeiro vou por o tamanho e depois o nome
 buf[1] = 0;
 buf[2] = sizeof(filesize);
 memcpy(buf+3, &filesize, sizeof(filesize));

 buf[3 +sizeof(filesize)] = 1;
 buf[5] = sizeof(filename);
 memcpy(buf + 4 +sizeof(filesize), &filesize, strlen(filename));



}

void comporPacotesDados(int seqNumb, int sizeCampoI, int lengthDados, char* dados){

 buf[0] = 0;
 buf[1] = seqNumb;
 buf[2] = lengthDados/256;
 buf[3] = lengthDados%256;
int i;
for(i=0; i < lengthDados; i++){

 buf[4] = dados[i];

}

}
```

**writenoncanonical.c**

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

#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define F 0x7E
#define A 0x03
#define C_SET 0x07
#define BCC (A^C_SET)
#define C_UA 0x03

volatile int STOP=FALSE;
volatile int flag=FALSE;
unsigned char SET[5];
unsigned char SET2[5];

enum state {START2, FLAG2, A2, C2,BCC2, STOP2};

int estado = START2;

int fd, res;
int tentativas = 0;
unsigned char UA[5];

void state_machine(int state, char signal){
	printf("estado antes: %d \n", estado);

	if (state == START2){
		if (signal == F){
			state = FLAG2;
			UA[0]=signal;
		}
	}
	else if (state == FLAG2){
		if (signal == F)
			state = FLAG2;
```

```
            else if (signal == A){
                state = A2;
                UA[1]=signal;
            }
            else
                state = START2;
        }
        else if (state == A2){
            if (signal == F){
                state = FLAG2;
            }
            else if (signal == C_UA){
                state = C2;
                UA[2]=signal;
            }
            else
                state = START2;
        }
        else if (state == C2){
            if (signal == F)
                state = FLAG2;
            else if (signal == (UA[1]^UA[2])){
                state = BCC2;
                UA[3]=signal;
            }
            else
                state = START2;
        }
        else if (state == BCC2){
            if (signal == F){
                state = STOP2;
                UA[4]=signal;
            }
            else
                state = START2;
        }
        estado = state;
        printf("estado após: %d \n", estado);

}

int confirmar(){

    char buf2;
    int res2;
    int ecx=0;

    while(ecx <5){
```

```
            ecx++;
            while( !(res2 = read(fd, &buf2, 1)) )
                    continue;
            printf("Received: %x !!! %d \n", buf2, res2);
            state_machine(estado, buf2);
            if(estado == STOP2)
                    return 1;

    }
            return 0;


}

void send_SET(){
            int i = 0;
    while(i < 5){
            res = write(fd,&SET[i],1);
            i++;
            }
            i=0;
            printf("Send: 0x%x 0x%x 0x%x 0x%x 0x%x \n", SET[0], SET[1], SET[2], SET[3],
SET[4]);
}

void atende() // atende alarme
{
            if(!flag){
                    send_SET();
            };
}

int main(int argc, char** argv)
{
    struct termios oldtio,newtio;
    char buf[255];

    (void) signal(SIGALRM, atende);

    if ( (argc < 2) ||
                ((strcmp("/dev/ttyS4", argv[1])!=0) &&
                 (strcmp("/dev/ttyS1", argv[1])!=0) )) {
      printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
      exit(1);
    }

    fd = open(argv[1], O_RDWR | O_NOCTTY );
    if (fd <0) {perror(argv[1]); exit(-1); }
```

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

  bzero(&newtio, sizeof(newtio));
  newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
  newtio.c_iflag = IGNPAR;
  newtio.c_oflag = OPOST;

  /* set input mode (non-canonical, no echo,...) */
  newtio.c_lflag = 0;

  newtio.c_cc[VTIME]    = 0;   /* inter-character timer unused */
  newtio.c_cc[VMIN]     = 0;   /* blocking read until 5 chars received */


  tcflush(fd, TCIFLUSH);

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

        SET[0] = F;
        SET[1] = A;
        SET[2] = C_SET;
        SET[3] = A^C_SET;
        SET[4] = F;

        send_SET();

        while(!confirmar() && tentativas < 3){
              alarm(3);
                 tentativas++;
                 send_SET();
        }


        int i = 0;
        flag = TRUE;

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

  sleep(5);
```

```
    if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
    }
    close(fd);
    return 0;
}
```