U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

MIEIC 3º Ano

RCOM

Novembro 2020

Serial Port: Protocolo de Ligação de Dados

Relatório

(1º Trabalho Laboratorial)

Turma 1 - Grupo 6

Rui Filipe Mendes Pinto - up201806441

Tiago Gonçalves Gomes – up201806658

Sumário

Este relatório foi elaborado no sentido de descrever a nossa implementação do primeiro trabalho laboratorial, cujo objetivo principal é implementar um protocolo de ligação de dados para uma aplicação capaz de transferir ficheiros de um computador para o outro através de uma porta de série. Todos os objetivos deste trabalho foram atingidos, ficando nós com uma aplicação perfeitamente funcional, mesmo quando é introduzido ruído na porta de série ou interrompida a ligação.

Introdução

O trabalho laboratorial visa a implementação de um protocolo de ligação de dados, recorrendo a uma porta de série. Através desta, são transferidos ficheiros numa ligação do tipo *Point-To-Point*, ou seja, uma comunicação entre duas *stations*; uma com o papel de emissora, a outra com o papel de recetora. Relativamente ao relatório, o seu objetivo é retratar todas as questões teóricas presentes neste primeiro trabalho.

Arquitetura

O programa está dividido em dois blocos funcionais: o emissor e o recetor. Cada um desses blocos incorpora a camada de ligação de dados e a camada de aplicação de uma forma isolada.

Estrutura de código

O código do programa está dividido em 7 ficheiros: 3 ficheiros .c e 4 ficheiros .h (3 headers dos ficheiros .c e 1 ficheiro com macros).

Writer

Funções principais da camada de ligação:

- **llopen –** configura a porta de série, envia a trama de supervisão SET e recebe a trama UA.
- **llwrite –** encapsula a mensagem (trama I) a enviar, realiza o cálculo do BCC2 e o *stuffing* das tramas I e envia-as. Seguidamente espera pela resposta do recetor e repete o processo.
- **llclose –** envia tramas de supervisão *DISC*, recebe *DISC*, envia *UA*.

Funções principais da camada de aplicação:

- **main –** base da camada de aplicação; faz as chamadas às funções da camada de ligação.
- **openReadFile –** abre um ficheiro e lê o seu conteúdo.
- **buildControlTrama –** constrói os pacotes de controlo com o tamanho do ficheiro e o nome do mesmo.
- **encapsulateMessage –** encapsula a mensagem a enviar para ficar de acordo com as tramas I.

Variáveis globais:

- **numRetries –** contador de *timeouts*, inicializado a 0.
- **readSuccessful –** *flag* que indica se conseguiu ler uma trama com sucesso, inicializada a *FALSE*.
- **fd –** *file descriptor* da porta de série.
- **toSend –** buffer que aloca a última trama I enviada. Utilizado para processos de reenvio da mesma.
- **finalIndex –** detém o número de bytes da a última trama I enviada. Utilizada no reenvio da mesma e na função llwrite.

Reader

Funções principais da camada de ligação:

- **llopen –** configura a porta de série, recebe a trama de supervisão *SET* e envia a trama *UA*.
- **llread –** desencapsula a mensagem (trama I) a receber, realiza o *destuffing* e o cálculo do BCC2 das tramas I. Seguidamente envia a resposta para o recetor.
- **llclose –** recebe tramas de supervisão *DISC*, envia *DISC*, recebe *UA*.

Funções principais da camada de aplicação:

- **main –** base da camada de aplicação; faz as chamadas às funções da camada de ligação.
- **checkEnd –** verifica se uma trama sinaliza o fim da transmissão (trama *END)* da mensagem.
- **sizeOfFile_Start –** obtém o tamanho do ficheiro a partir da trama *START*.
- **nameOfFile_Start –** obtém o nome do ficheiro a partir da trama *START*.
- **createFile –** cria ficheiro com os *bytes* recebidos nas tramas I.

Macros

Ficheiro com as macros utilizadas no programa

- **N_BYTES_TO_SEND –** comprimento em bytes da informação contida nos pacotes de dados.
- **MAX_RETR –** número máximo de tentativas de envio de uma trama.
- **TIMEOUT –** número de segundos do *timeout*.
- **BAUDRATE –** Capacidade de ligação da porta de série.
- **MAX_NUM_OCTETS** – simboliza o número máximo de bytes que um pacote de dados pode conter (2^{16} - associado aos campos *L2* e *L1*).
- **ERROR_MODE –** *flag* utilizada para determinar se estamos em modo de geração de erros ou variação do tempo de propagação de uma trama I.
- **BCC1_ERROR_PERCENTAGE –** percentagem de erros gerados no *BCC1*.
- **BCC2_ERROR_PERCENTAGE –** percentagem de erros gerados no *BCC2*.
- **T_PROP_ERROR_PERCENTAGE –** percentagem de erros associados à variação do tempo de propagação de uma trama I.

Casos de uso principais

Os principais casos de uso desta aplicação são: a interface, que permite ao transmissor escolher o ficheiro a enviar (através de um argumento da linha de comandos), e a transferência desse mesmo ficheiro, recorrendo à porta de série, entre dois computadores, o emissor e o recetor. Para iniciar a aplicação, o utilizador deverá inserir um conjunto de argumentos, utilizando a linha de comandos do Linux. Se for o emissor, deverá inserir qual a porta de série a ser utilizada (por exemplo, **/dev/ttyS0**), e o ficheiro a ser enviado (por exemplo, **pinguim.gif**). Sendo o recetor, basta inserir a porta de série. A transmissão dos dados que compõe o ficheiro dá-se com a seguinte sequência:

1. Emissor escolhe o ficheiro a enviar, na interface.
2. Configuração da ligação entre os dois computadores.
3. Estabelecimento da ligação.
4. Transmissor envia os dados.
5. Recetor recebe os dados.
6. Recetor guarda os dados num ficheiro com o mesmo nome do ficheiro enviado pelo emissor.
7. Terminação da ligação.

Protocolo de Ligação Lógica

O protocolo de Ligação Lógica tem como objetivo fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio (canal) de transmissão - neste caso, um cabo de série. O protocolo implementado tem como principais aspetos funcionais:

- Configuração da porta de série;
- Estabelecimento de ligação pela porta de série;
- Transferência de dados pela porta de série, fazendo *stuffing* e *destuffing* dos mesmos;
- Recuperação de erros durante a transferência de dados.

Para o mesmo, foi necessário implementar as seguintes funções:

```
void llopen(struct termios* oldtio, struct termios* newtio);
```

Esta função tem a responsabilidade de estabelecer a ligação entre o emissor e o recetor. No emissor, após a configuração da porta de série, é enviada uma trama SET, sendo nesse mesmo instante ativado um temporizador para reenvio do *SET* que é desativado no recebimento da resposta (*UA*). Este processo de retransmissão só é repetido um número máximo de vezes e recorre à utilização de um alarme (função ***setHandler***) . No recetor tem uma funcionalidade semelhante. Para enviar tanto o UA como o SET, é usada a função ***sendSupervisionTrama*** que recebe como argumentos o file descriptor (*fd*) da porta de série, o parâmetro *C* (Campo de Controlo) e o parâmetro *A* (Campo de Endereço) presentes nas tramas de Supervisão.. O mecanismo de receção do *UA* e do *SET*, requer a utilização de uma *state machine*, de modo a permitir a verificação de tramas incorretas que se encontra na função receiveSupervisionTrama que recebe os mesmos parâmetros da função ***sendSupervisionTrama*** e uma flag que sinaliza a ativação de um time-out necessário para reenvio da trama anteriormente enviada.

```
void llwrite(unsigned char *buf, off_t size, bool* nTrama);
```

Esta função é responsável pelo envio das tramas I, após o *stuffing* das mesmas, no emissor. Inicialmente é feito o *framing* da mensagem, ou seja, adicionado o cabeçalho presente no Protocolo de Ligação, à mensagem. A mesma poderá ter um tamanho variável de acordo com a macro ***N_BYTES_TO_SEND***. Em seguida, calcula-se o *BCC2* (função ***computeBcc2***), faz-se o stuffing da mensagem e do *BCC2* (função ***byteStuffing***) e procede-se à sua escrita. No caso do ***ERROR_MODE*** estar ativo, a função ***varyBCC1_BCC2*** é chamada, de modo a introduzir erros no *BCC1* e/ou *BCC2* da trama. Esta função substitui o conteúdo de uma posição aleatória (no caso do *BCC1* da posição 1 a 3 e no caso do *BCC2* nas posições ocupadas pelo campo de dados e *BCC2*) com uma letra maiúscula aleatória, de acordo com a probabilidade de erro escolhida.

O envio da trama tem também implementado o mecanismo de *time-out*, muito semelhante ao do *SET*, permitindo a retransmissão da trama I (função ***resendTrama***). O mesmo só é ativado caso não seja recebida uma resposta ***RR*** (emissor pode passar à transmissão da próxima trama I) ou ***REJ*** (recetor pediu a retransmissão da última trama enviada) na função ***receiveSupervisionTrama*** que, de acordo com o parâmetro C recebido, permite distinguir qual foi o tipo de mensagem enviada.

```
struct readReturn llread(int fd, int* tNumber);
```

Esta função é responsável pela receção das tramas I e pelo *destuffing* das mesmas, do lado do recetor. A leitura é feita caracter a caracter recorrendo à função ***receiveTrama***, onde está implementada a *state machine* que permite fazer o controlo da receção da mensagem. Em seguida procede-se ao *destuffing* da trama na função ***deStuffing***. Para verificar o *BCC2* recorre-se novamente à função ***computeBcc2***; caso esteja correto, é enviado um *RR*, caso contrário um *REJ*, usando a função ***sendSupervisionTrama***. O campo de controlo enviado depende do número de sequência da trama (Nr), presente na variável ***tNumber***. No meio de todo este processo é guardado o campo de Controlo recebido para posterior comparação com o valor de *C* esperado (Ns). No caso de serem diferentes, significa que a trama é duplicada e, portanto, não será guardada. Esta função retorna uma *struct* que contém um *unsigned char ** para a mensagem lida, bem como um *int* que detém o comprimento da mesma.

```
void llclose(int fd);
```

Esta função tem a responsabilidade de terminar a ligação entre o emissor e o recetor. No emissor, é enviada a trama da Supervisão *DISC*, através da função ***sendSupervisionTrama*** e espera outro *DISC* na função ***receiveSupervisionTrama***. Por fim, envia um *UA*, terminando assim a ligação. No lado do recetor, é esperado um *DISC*, enviado também um *DISC* e esperado um *UA*, também através das funções ***receiveSupervisionTrama*** e ***sendSupervisionTrama***.

Protocolo de Aplicação

O protocolo de Aplicação implementado tem como aspetos principais:

- Envio de pacotes de controlo *START* and *END*, que contêm o nome e tamanho do ficheiro a ser enviado;
- Encapsulamento de cada fragmento de dados da mensagem antes de a enviar, por parte do emissor e concatenação dos fragmentos recebidos, por parte do receptor;
- Leitura do ficheiro a enviar por parte do emissor, e criação do ficheiro por parte do recetor.

Para o mesmo, foi necessário implementar as seguintes funções:

```
unsigned char* openReadFile(char* fileName, off_t* fileSize);
```

Esta função, presente no emissor, abre e lê um ficheiro, com o nome *filename*, retornando o seu conteúdo e, como argumento, o tamanho do ficheiro *fileSize*.

```
unsigned char* buildControlTrama(char* fileName, off_t* fileSize, unsigned char controlField, off_t* packageSize);
```

Esta função, presente no emissor, constrói os pacotes de controlo com o tamanho do ficheiro e o nome do mesmo. Estas tramas indicam o início (pacotes de controlo *START* e *END*) e o fim da transmissão / receção de dados, através do campo *C*.

```
unsigned char* encapsulateMessage(unsigned char* messageToSend, off_t* messageSize, off_t* totalMessage, int* numPackets);
```

Esta função, presente no emissor, encapsula a mensagem a enviar, de maneira a ficar de acordo com os pacotes de dados, contendo o número de sequência do pacote (módulo 255) e o tamanho do fragmento.

```
bool checkEnd(unsigned char* endMessage, off_t endMessageSize, unsigned char* startMessage, off_t startMessageSize);
```

Esta função, presente no recetor, verifica se a mensagem atual corresponde à trama de controlo *END*, de maneira a podermos parar a troca de informação entre o emissor e o recetor.

```
off_t sizeOfFile_Start(unsigned char *start);
```

Esta função, presente no emissor, obtém o tamanho do ficheiro a partir da trama de controlo *START*.

```
unsigned char *nameOfFile_Start(unsigned char *start);
```

Esta função, presente no emissor, obtém o nome do ficheiro a partir da trama de controlo *START*.

```
void createFile(unsigned char *data, off_t* sizeFile, unsigned char filename[]);
```

Esta função, presente no recetor, cria um ficheiro com com a informação recebida nos pacotes de dados. O ficheiro criado tem o mesmo nome do ficheiro transmitido, obtido através da trama de controlo *START*.

Validação

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

- Envio de ficheiros de vários tamanhos (desde 11kB até 200kB, na porta de série presente no laboratório, e até 50MB, na porta de série virtual do *socat*).
- Introdução de ruído e geração de curto circuito na porta de série enquanto se envia o ficheiro.
- Interrupção da ligação durante alguns segundos (até 9 segundos, já que o tempo do *timout* é 3 segundos e o número de tentativas é 3) enquanto se envia o ficheiro.
- Variação do tempo de propagação de 1s na receção de cada trama (probabilidades compreendidas entre 0 e 25%).
- Envio de um ficheiro com variação na percentagem de erros simulados (desde 0% até 6%).
- Envio de um ficheiro com variação no tamanho de pacotes (pacotes com tamanho desde os 20 bytes até aos 500 bytes).
- Envio de um ficheiro com variação das capacidades de ligação (*baudrate* [desde 2400 até 38400]).

Eficiência do protocolo de ligação de dados

De forma a avaliar a eficiência do protocolo desenvolvido, foram feitos alguns testes. Procedeu-se sempre à realização de dois testes nas mesmas condições para depois se utilizar a sua média, de modo a reduzir o desvio dos dados. Todas as tabelas estão presentes no anexo II.

Variação do FER

Este gráfico permite concluir que a geração de erros no *BCC1* e no *BCC2* tem um grande impacto na eficiência do programa. Isto acontece porque quando há erros no *BCC1* tem de ser aplicado o reenvio da trama havendo associada a essa operação um *time-out* inerente. Os erros no *BCC2* não têm tanto efeito pois só causa o reenvio imediato da trama. O gráfico encontra-se na figura 1 do anexo II.

Variação do tamanho das tramas I

Este gráfico permite confirmar que quanto maior o tamanho de cada pacote, mais eficiente é a aplicação. Isto devido ao facto de ser enviada mais informação fazendo com que menos tramas sejam mandadas e o programa execute mais rapidamente. O gráfico encontra-se na figura 2 do anexo II.

Variação da Capacidade de Ligação (C)

Este gráfico permite concluir que o aumento da capacidade de ligação, diminui a eficiência. O gráfico encontra-se na figura 3 do anexo II.

Variação do Tempo de Propagação

Este gráfico permite concluir que à medida que a percentagem de erro associado ao tempo de propagação aumenta, a eficiência diminui. O gráfico encontra-se na figura 4 do anexo II.

Relativamente ao protocolo *Stop & Wait*, após a transmissão de um pacote de informação, o emissor aguarda pela confirmação de recebimento da trama por parte do receptor (*acknowledgment* - **ACK**). Quando o receptor recebe o pacote, no caso de não existir qualquer erro, confirma com **ACK**, caso contrário envia **NACK**. No momento em que o emissor recebe a resposta do receptor, no caso de **ACK**, continua e envia um novo pacote, no caso de **NACK**, volta a enviar o mesmo pacote.

Na aplicação desenvolvida foi usado o protocolo *Stop & Wait*. Quando o emissor manda uma trama (U, S ou I) espera sempre uma resposta. Essa resposta é **RR** caso o recetor receba os dados sem erros, e **REJ** no caso contrário. Deste modo, o emissor sabe sempre se pode enviar uma nova trama ou se tem de reenviar a anterior. Para garantir este processo, o *Nr* das tramas de resposta varia conforme o emissor tenha enviado a trama de *Ns* 0 ou 1, de modo a que o mesmo consiga saber que trama deve enviar a seguir e para ajudar no tratamento de duplicados.

Conclusões

Durante as últimas semanas, o grupo teve em mãos o desenvolvimento de uma aplicação capaz de transferir ficheiros entre dois computadores através de uma ligação através da porta de série.

Durante as duas primeiras aulas, foram-nos fornecidos guiões, que nos serviram para guiar num momento em que ainda estávamos um pouco 'perdidos', pelo que foram de grande utilidade. É bom salientar também que os docentes da unidade curricular foram exemplares, tanto na exposição da parte teórica, como no esclarecimento de dúvidas, quer em sala de aula, quer por email.

Um termo importante que retemos deste trabalho é o conceito de independência entre camadas, no caso do nosso programa, a independência entre a camada de ligação de dados e a camada da aplicação. Na camada de ligação de dados não é feito qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de informação. Quanto à camada de aplicação, esta não conhece os detalhes do protocolo de ligação de dados, mas a forma como o serviço é acedido.

Após a finalização, o grupo conclui que todos os objetivos relacionados com a estrutura e robustez do trabalho foram atingidos, e que vários conceitos e aspetos relativos ao tema em questão, adquiridos em aulas teóricas, foram aprofundados.

Anexo I

Writer.h:

```
#pragma once

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

#include "generalFunctions.h"

/*-------------------------- Data Link Layer --------------------------*/

/*
* Aciona o Timeout da trama de controlo SET.
* Data Link Layer
*/
void setHandler(int sigNum);

/*
* Aciona o Timeout da trama de controlo DISC.
* Data Link Layer
*/
void resendDISC(int signum);

/*
* Aciona o Timeout da trama I.
* Data Link Layer
*/
void resendTrama(int sigNum);

/*
* Faz o stuffing dos bytes de informação.
* Data Link Layer
*/
int byteStuffing(unsigned char* dataToSend, int size);

/*
* Configura a porta de série, envia a trama de controlo SET e recebe a trama UA.
* Data Link Layer
*/
void llopen(struct termios* oldtio, struct termios* newtio);
```

```
/*
* Envia as tramas I.
* Data link layer
*/
void llwrite(unsigned char *buf, off_t size, bool* nTrama);

/*
* Envia a trama de controlo DISC, recebe DISC de volta e envia UA.
* Data link Layer
*/
void llclose();

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

/*
* Abre um ficheiro e lê o seu conteúdo.
* Application layer
*/
unsigned char* openReadFile(char* fileName, off_t* fileSize);

/*
* Constrói os pacotes de controlo com o tamanho do ficheiro e o nome do mesmo.
* Application layer
*/
unsigned char* buildControlTrama(char* fileName, off_t* fileSize, unsigned char controlField,
off_t* packageSize);

/*
* Encapsula a mensagem a enviar para ficar de acordo com os pacotes de dados.
* Application layer
*/
unsigned char* encapsulateMessage(unsigned char* messageToSend, off_t* messageSize, off_t*
totalMessageSize, int* numPackets);
```

Reader.h:

```
#pragma once

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

#include "generalFunctions.h"

/*-------------------------- Data Link Layer --------------------------*/
```

```
/*
* Faz o destuffing de uma mensagem.
* Data Link Layer
*/
struct readReturn deStuffing(unsigned char * message, int size);

/*
* Configura a porta de série, lê a trama de controlo SET e envia a trama UA.
* Data Link Layer
*/
void llopen(int fd, struct termios* oldtio, struct termios* newtio);

/*
* Lê tramas I.
* Data Link Layer
*/
struct readReturn llread(int fd, int* tNumber);

/*
* Recebe os diferentes bytes da trama.
* Data Link Layer
*/
struct receiveTramaReturn receiveTrama(int* nTrama, int fd);

/*
* Lê trama de controlo DISC, envia DISC de volta e recebe UA.
* Data link Layer
*/
void llclose(int fd);

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

/*
* Verifica se é a última mensagem.
* Application Layer
*/
bool checkEnd(unsigned char* endMessage, off_t endMessageSize, unsigned char*
startMessage, off_t startMessageSize);

/*
* Obtém o tamanho do ficheiro a partir da trama START.
* Application Layer
*/
off_t sizeOfFile_Start(unsigned char *start);

/*
* Obtém o nome do ficheiro a partir da trama START.
* Application Layer
*/
unsigned char *nameOfFile_Start(unsigned char *start);

/*
* Cria ficheiro com os dados recebidos nas tramas I.
* Application Layer
*/
```

```
void createFile(unsigned char *data, off_t* sizeFile, unsigned char filename[]);
```

GeneralFunctions.h:

```
#pragma once

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

#include "macros.h"

/*
* Retorna o Campo de Controlo (C) especificado.
*/
unsigned char getCField(char typeMessage[25], bool nTrama);

/*
* Faz o parse do BCC2 de acordo com o pacote de dados recebido.
*/
unsigned char computeBcc2(unsigned char data[], int nBytes, int startPosition);

/*
*    Open serial port device for reading and writing and not as controlling tty
*     because we don't want to get killed if linenoise sends CTRL-C.
*/
void configureSerialPort(int fd, struct termios* oldtio, struct termios* newtio);

/*
* Lê uma trama de Supervisão.
*/
int receiveSupervisionTrama(bool withTimeout, unsigned char cField, int fd, unsigned char
aField);

/*
* Envia uma trama de Supervisão.
*/
void sendSupervisionTrama(int fd, unsigned char cField, unsigned char aField);

/*
* Verifica se o número máximo de Bytes a enviar foi excedido (2^16)
*/
void checkMaxBytesToSend(off_t* packageSize);
```

```
struct readReturn {
   unsigned char* currentMessage;
   int currentMessageSize;
};

struct receiveTramaReturn {
   unsigned char* currentMessage;
   int statusCode;
   int currentMessageSize;
};
```

Macros.h:

```
#define FLAG_SET 0X7E
#define A_C_SET 0x03
#define Other_A 0x01

#define C_I( n ) ( (n == 0) ? 0x00 : 0x40 )

#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

#define N_BYTES_FLAGS 6 // Número de bytes pertencentes a FLAGS na Trama de Informação
#define N_BYTES_TO_SEND 128

#define MAX_RETR 3
#define TIMEOUT 3

#define C_SET 0x03
#define C_DISC 0x0B
#define C_UA 0x07
#define C_RR( n )  ( (n) ? 0x85 : 0x05 )
#define C_REJ( n ) ( (n) ? 0x81 : 0x01 )

#define ESC 0x7D
#define ESC_XOR1 0X5E
#define ESC_XOR2 0x5D

#define DATA_HEADER_LEN 4
#define MODULE 256

#define C_DATA 0x01
#define C_START 0x02
#define C_END 0x03
#define T1 0x00
#define T2 0x01
#define L1 0x04
```

```
#define MAX_NUM_OCTETS 65536 // 2^16

#define MIN(a, b) ( (a < b) ? a : b)

#define ERROR_MODE 0
// ERROR_MODE:
//     0. Non Error Mode
//     1. Vary T_prop
//     2. Vary FER

#define BCC2_ERROR_PERCENTAGE 5
#define BCC1_ERROR_PERCENTAGE 5
#define T_PROP_ERROR_PERCENTAGE 5
```

Writer.c:

```
/*Non-Canonical Input Processing*/

#include "headers/writer.h"

int numRetries = 0;

bool readSuccessful = false;
int fd;

unsigned char toSend[N_BYTES_FLAGS + (N_BYTES_TO_SEND * 2)]; // Buffer da mensagem a
enviar
int finalIndex;

void setHandler(int sigNum) {

  if(numRetries < MAX_RETR) {
    if(readSuccessful)
      return;

    printf("\n--- Sending SET: Retry %d ---\n", numRetries);
    sendSupervisionTrama(fd, getCField("SET", false), A_C_SET);
    alarm(TIMEOUT);
    numRetries++;
  }
  else {
    printf("Can't connect to Receptor! (After %d tries)\n", MAX_RETR);
    exit(1);
  }
}

void resendDISC(int signum) {
  if(numRetries < MAX_RETR) {
    if(readSuccessful)
```

```
    return;

   printf("\n--- Sending DISC: Retry %d ---\n", numRetries);
   sendSupervisionTrama(fd, getCField("DISC", false), A_C_SET);
   alarm(TIMEOUT);
   numRetries++;
  }
  else {
   printf("Can't connect to Receptor! (After %d tries)\n", MAX_RETR);
   exit(1);
  }
}

void resendTrama(int sigNum) {
  if(numRetries < MAX_RETR) {
   if(readSuccessful)
    return;

   printf("\nSending Trama: Retry %d\n", numRetries);
   write(fd, toSend, finalIndex + 1);

   alarm(TIMEOUT);
   numRetries++;
  }
  else {
   printf("After %d tries to resend trama, it didn't work. Exiting program!\n", MAX_RETR);
   exit(1);
  }
}

int byteStuffing(unsigned char* dataToSend, int size) {
  int pos = 0, startingPos = 4;
  int index = 0, bytesTransformed = 0;

  unsigned char auxToSend[N_BYTES_TO_SEND * 2];

  while(index < size) {
   if(dataToSend[startingPos] == FLAG_SET) { // Há mais casos -> Slide 13
    auxToSend[pos++] = ESC;
    auxToSend[pos] = ESC_XOR1;
    bytesTransformed++;
   }
   else if (dataToSend[startingPos] == ESC) {
    auxToSend[pos++] = ESC;
    auxToSend[pos] = ESC_XOR2;
    bytesTransformed++;
   }
   else
    auxToSend[pos] = dataToSend[startingPos];

   bytesTransformed++;
   pos++;
   startingPos++;
   index++;
  }
```

```
  // Copy "stuffed" array into toSend
  memcpy(&toSend[4], auxToSend, bytesTransformed);
  return 4 + bytesTransformed;
}

void llopen(struct termios* oldtio, struct termios* newtio) {
  configureSerialPort(fd, oldtio, newtio);

  // SEND SET
  printf("\n--- Sending SET ---\n");
  sendSupervisionTrama(fd, getCField("SET", false), A_C_SET);

  // RECEIVE UA
  readSuccessful = receiveSupervisionTrama(true, getCField("UA", false), fd, A_C_SET) == 1;
  printf("\n--- RECEIVED UA ---\n");

}

void varyBCC1_BCC2(unsigned char* message, int messageSize, bool* nTrama) {
  if(ERROR_MODE == 2) {
    unsigned char *copyToSend = (unsigned char *) malloc (messageSize);
    memcpy(copyToSend, message, messageSize);
    // BCC2
    int r = (rand() % 100) + 1;
    if(r <= BCC2_ERROR_PERCENTAGE) {
      int i = (rand() % (messageSize - 5)) + 4; // We don't want to change the FLAG
      unsigned char randomLetter = (unsigned char) ('A' + (rand() % 26));
      copyToSend[i] = randomLetter;
      printf("\n-- BCC2 Changed! --\n");
    }

    // BCC1
    r = (rand() % 100) + 1;
    if(r <= BCC1_ERROR_PERCENTAGE) {
      int i = (rand() % 3) + 1;
      unsigned char randomLetter = (unsigned char) ('A' + (rand() % 26));
      copyToSend[i] = randomLetter;
      printf("\n-- BCC1 Changed! --\n");
    }

    printf("\nSENT TRAMA (%d)!\n", *nTrama);
    write(fd, copyToSend, messageSize);
    free(copyToSend);
  }
}

void llwrite(unsigned char *buf, off_t size, bool* nTrama) {
    int i = 0;
    int j;
    int nTramasSent = 0;
    int nBytesRead = 0;

    while(i < size) {
      j = 4;

      toSend[0] = FLAG_SET; // F
```

```
    toSend[1] = A_C_SET; // A
    toSend[2] = C_I(*nTrama); // C
    toSend[3] = A_C_SET ^ toSend[2]; // BCC1

    nBytesRead = 0;
    while(nBytesRead < N_BYTES_TO_SEND && i < size) {
     toSend[j] = buf[i];

     nBytesRead++;
     i++;
     j++;
    }

    toSend[j] = computeBcc2(buf, nBytesRead, nTramasSent * N_BYTES_TO_SEND);
    finalIndex = byteStuffing(toSend, nBytesRead + 1); // +1 because of BCC2
    toSend[finalIndex] = FLAG_SET;

    if(ERROR_MODE == 2)
     varyBCC1_BCC2(toSend, finalIndex + 1, nTrama);
    else {
     printf("\nSENT TRAMA (%d)!\n", *nTrama);
     write(fd, toSend, finalIndex + 1);
    };

    readSuccessful = false;
    int returnState = receiveSupervisionTrama(true, getCField("RR", !(*nTrama)), fd, A_C_SET); //
[Nr = 0 | 1]
    readSuccessful = true;
    numRetries = 0;

    if(returnState == 1) {
     printf("\nReceiving RR\n");
     nTramasSent++;
     *nTrama = !(*nTrama);
    }
    else if(returnState == 2) { // Retransmissão da última trama enviada
     printf("\nReceiving REJ\n");
     i -= nBytesRead;
    }
   }
}

unsigned char* openReadFile(char* fileName, off_t* fileSize) {
 FILE* f;
 struct stat metadata;
 unsigned char* fileData;

 if((f = fopen(fileName, "rb")) == NULL) {
  perror("Error opening file!");
  exit(-1);
 }
 // off_t st_size
 stat(fileName, &metadata);
 *fileSize = metadata.st_size;
 printf("File %s has %ld bytes!\n", fileName, *fileSize);
```

```
  fileData = (unsigned char *) malloc(*fileSize); // fileData will hold the file bytes -> DON'T
FORGET TO FREE MEMORY AT THE END!
  fread(fileData, sizeof(unsigned char), *fileSize, f);

  return fileData;
}

unsigned char* buildControlTrama(char* fileName, off_t* fileSize, unsigned char controlField,
off_t* packageSize) {
  int fileNameSize = strlen(fileName); // Size in bytes (chars)
  int sizeControlPackage = 5 * sizeof(unsigned char) + L1 * sizeof(unsigned char) + fileNameSize *
sizeof(unsigned char);

  unsigned char* package = (unsigned char*) malloc(sizeControlPackage);
  package[0] = controlField;
  package[1] = T1;
  package[2] = L1;
  package[3] = ((*fileSize) >> 24) & 0xFF;
  package[4] = ((*fileSize) >> 16) & 0xFF;
  package[5] = ((*fileSize) >> 8) & 0xFF;
  package[6] = ((*fileSize)) & 0xFF;
  package[7] = T2;
  package[8] = fileNameSize;

  for(int i = 0; i < fileNameSize; i++)
    package[9 + i] = fileName[i];

  *packageSize = 9 + fileNameSize;

  checkMaxBytesToSend(packageSize);

  return package;
}

unsigned char* encapsulateMessage(unsigned char* messageToSend, off_t* messageSize, off_t*
totalMessageSize, int* numPackets) {
  unsigned char* totalMessage = (unsigned char*) malloc(0);

  int16_t aux;
  off_t auxBytesToSend = 0;
  int8_t sequenceNum = 0;

  for(off_t i = 0; i < *messageSize; i++) {
    if(i == auxBytesToSend) { // Each packet has N_BYTES_TO_SEND bytes of Info
      totalMessage = (unsigned char *) realloc(totalMessage, *totalMessageSize +
DATA_HEADER_LEN);
      totalMessage[(*totalMessageSize)++] = C_DATA;                    // C
      totalMessage[(*totalMessageSize)++] = sequenceNum;               // N
      aux = MIN((N_BYTES_TO_SEND - DATA_HEADER_LEN), (*messageSize - i)); // MIN(Number of
Bytes we can send, remaining bytes)
      totalMessage[(*totalMessageSize)++] = (aux >> 8) & 0xFF; // L2
      totalMessage[(*totalMessageSize)++] = (aux) & 0xFF;      // L1

      sequenceNum = (sequenceNum + 1) % MODULE; // [0 - 255]
      auxBytesToSend += aux;
      (*numPackets)++;
```

```
    }
    totalMessage = (unsigned char *) realloc(totalMessage, *totalMessageSize + 1);
    totalMessage[(*totalMessageSize)++] = messageToSend[i];
  }

  free(messageToSend);

  return totalMessage;
}

void llclose() {
  printf("\n-- SENT DISC --\n");
  sendSupervisionTrama(fd, getCField("DISC", true), A_C_SET);
  (void) signal(SIGALRM, resendDISC);
  numRetries = 0;
  readSuccessful = false;
  receiveSupervisionTrama(true, getCField("DISC", true), fd, Other_A);
  printf("\n-- RECEIVED DISC --\n");
  readSuccessful = true;
  printf("\n-- SENT UA --\n");
  sendSupervisionTrama(fd, getCField("UA", true), Other_A);
}

int main(int argc, char** argv)
{
  srand(time(NULL));
  (void) signal(SIGALRM, setHandler);
  bool nTrama = false; // [Ns = 0 | 1]

  struct termios oldtio,newtio;

  if ( (argc < 3) ||
       ((strcmp("/dev/ttyS10", argv[1])!=0) &&
       (strcmp("/dev/ttyS0", argv[1])!=0) )) {
    printf("Usage:\tnserial SerialPort filename\n\tex: nserial /dev/ttyS1 pinguim.gif\n");
    exit(1);
  }

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

  off_t fileSize;
  unsigned char *msg = openReadFile(argv[2], &fileSize);

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

  llopen(&oldtio, &newtio);

  (void) signal(SIGALRM, resendTrama); // Registering a new SIGALRM handler

  // START -> Construct
  off_t startPackageSize = 0;
  unsigned char* startPackage = buildControlTrama(argv[2], &fileSize, C_START,
&startPackageSize);
  printf("\nSTART Frame size: %ld\n", startPackageSize);
```

```
 // START -> Send
 printf("\n-- SENT START --\n");
 llwrite(startPackage, startPackageSize, &nTrama);

 numRetries = 0;

 unsigned char* totalMessage;
 off_t totalMessageSize = 0;
 int numPackets;
 totalMessage = encapsulateMessage(msg, &fileSize, &totalMessageSize, &numPackets);

 // Trama de Informação para o Recetor

 printf("numPackets to Send: %d - \n", numPackets);
 llwrite(totalMessage, totalMessageSize, &nTrama);

 free(totalMessage);

 // END -> Construct
 off_t endPackageSize = 0;
 unsigned char* endPackage = buildControlTrama(argv[2], &fileSize, C_END, &endPackageSize);
 printf("\nEND Frame size: %ld\n", endPackageSize);

 // END -> Send
 printf("\n-- SENT END --\n");
 llwrite(endPackage, endPackageSize, &nTrama);

 llclose();

 printf("\nEND!\n");

 sleep(1);

 if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
   perror("tcsetattr");
   exit(-1);
 }
 struct timespec endTime;
 clock_gettime(CLOCK_REALTIME, &endTime);
 double sTime = startTime.tv_sec + startTime.tv_nsec * 1e-9;
 double eTime = endTime.tv_sec + endTime.tv_nsec * 1e-9;
 printf("-Time Passed: %.6lf-\n", eTime - sTime);

 close(fd);
 return 0;
}


// HOW TO COMPILE!
// gcc noncanonical.c generalFunctions.c -o nC
// gcc writenoncanonical.c generalFunctions.c -o writeNC
```

Reader.c:

```
/*Non-Canonical Input Processing*/

#include "headers/reader.h"

struct readReturn deStuffing(unsigned char * message, int size) {
  int currentMessageSize = 0;
  unsigned char* dataBytes = (unsigned char*) malloc(0);

  struct readReturn returnStruct;

  for(int i = 0; i < size; i++) {
    dataBytes = (unsigned char*) realloc(dataBytes, currentMessageSize + 1);

    if(message[i] == ESC) {
      if(i + 1 < size) {
        if(message[i+1] == ESC_XOR1) {
          dataBytes[currentMessageSize++] = FLAG_SET;
          i++;
        } else if(message[i+1] == ESC_XOR2) {
          dataBytes[currentMessageSize++] = ESC;
          i++;
        }
      } else {
        printf("\n--- String Malformed (Incorrect Stuffing)! ---\n");
        returnStruct.currentMessage = dataBytes;
      returnStruct.currentMessageSize = -1;
      return returnStruct;
      }
    } else if(message[i] == FLAG_SET){
      printf("\n--- String Malformed (Incorrect Stuffing)! ---\n");
      returnStruct.currentMessage = dataBytes;
      returnStruct.currentMessageSize = -1;
      return returnStruct;
    } else {
      dataBytes[currentMessageSize++] = message[i];
    }

  }

  memset(message, 0, size); // Clear array
  memcpy(message, dataBytes, currentMessageSize); // Substitute original array with destuffed
content

  returnStruct.currentMessage = dataBytes;
  returnStruct.currentMessageSize = currentMessageSize;
  // free(dataBytes);

  return returnStruct;
}

void varyT_prop() {
  if(ERROR_MODE == 1) { // Vary T_Prop
```

```
   int r = (rand() % 100) + 1;
   if(r <= T_PROP_ERROR_PERCENTAGE) {
     sleep(1);
     printf("\nT_Prop changed!\n");
   }
  }
}

struct receiveTramaReturn receiveTrama(int* nTrama, int fd) {
  varyT_prop();

  unsigned char buf;
  char state[6][25] = { "START", "FLAG_RCV", "A_RCV", "C_RCV", "BCC1_OK", "STOP" };
  int i = 0;
  bool repeatedByte = false;

  bool cFlag; // !nTrama = [Ns = 0 | 1]

  unsigned char* dataBytes = (unsigned char*) malloc(0);
  int index = 0;

  while (strcmp(state[i], "STOP") != 0) {      /* loop for input */

    read(fd, &buf, 1);   /* returns after 1 chars have been input */

    // Special cases (as they use dynamic fields)
    if(buf == (A_C_SET ^ C_I(*nTrama)) || buf == (A_C_SET ^ C_I(!(*nTrama)))) { // BCC1
      if(strcmp(state[i], "C_RCV") == 0) {
        i++;
        continue;
      }
    }
    else if(buf == C_I(0)) { // C
      if(strcmp(state[i], "A_RCV") == 0) {
        cFlag = 0;
        i++;
        continue;
      }
    }
    else if(buf == C_I(1)) { // C
      if(strcmp(state[i], "A_RCV") == 0) {
        cFlag = 1;
        i++;
        continue;
      }
    }

    switch (buf)
    {
      case FLAG_SET:
        if(strcmp(state[i], "START") == 0 || strcmp(state[i], "BCC1_OK") == 0)
          i++;
        else if(strcmp(state[i], "BCC1_OK") != 0)
          i = 1; // STATE = FLAG_RCV
        break;
      case A_C_SET: // A
```

```
      if(strcmp(state[i], "FLAG_RCV") == 0)
        i++;
      else if(strcmp(state[i], "BCC1_OK") != 0)
        i = 0; // Other_RCV
      else {
        dataBytes = (unsigned char*) realloc(dataBytes, index + 1);
        dataBytes[index++] = buf;
      }
      break;

    default:
      if(strcmp(state[i], "BCC1_OK") == 0) {
        dataBytes = (unsigned char*) realloc(dataBytes, index + 1);
        dataBytes[index++] = buf;
      }
      else // Other_RCV
        i = 0;
   }
  }

  if(cFlag == *nTrama) // repeatedByte
    repeatedByte = true;

  // Destuffing - Including BCC2

  struct receiveTramaReturn receiveTramaRet;

  struct readReturn deStuffingRet = deStuffing(dataBytes, index);
  receiveTramaRet.currentMessage = deStuffingRet.currentMessage;
  receiveTramaRet.currentMessageSize = deStuffingRet.currentMessageSize;

  if(deStuffingRet.currentMessageSize == -1) {
    receiveTramaRet.statusCode = -1;
    return receiveTramaRet; // PEDIR RETRANSMISSÃO (REJ)
  }

  // At this point dataBytes[currentMessageSize - 1] holds BCC2
  unsigned char bcc2 = computeBcc2(deStuffingRet.currentMessage,
deStuffingRet.currentMessageSize - 1, 0); // Excluding BCC2
  free(dataBytes);


  if(bcc2 != deStuffingRet.currentMessage[deStuffingRet.currentMessageSize - 1]) {
    if(repeatedByte) {
      *nTrama = cFlag;
      receiveTramaRet.statusCode = 2;
      return receiveTramaRet; // Status Code for Repeated Byte -> Descartar campo de dados
    }
    else {
      receiveTramaRet.statusCode = -1;
      return receiveTramaRet; // Status Code Error -> PEDIR RETRANSMISSÃO (REJ)
    }
  }
  *nTrama = cFlag;
  if(repeatedByte) {
    receiveTramaRet.statusCode = 2;
```

```
    return receiveTramaRet; // Status Code for Repeated Byte -> Descartar campo de dados
  }
  receiveTramaRet.statusCode = 0;
  return receiveTramaRet;
}

void llopen(int fd, struct termios* oldtio, struct termios* newtio) {
  configureSerialPort(fd, oldtio, newtio);

  // RECEIVE SET
  receiveSupervisionTrama(false, getCField("SET", false), fd, A_C_SET);
  printf("\n--- RECEIVING SET ---\n");

  // SEND UA
  sendSupervisionTrama(fd, getCField("UA", false), A_C_SET);
  printf("\n--- SENDING UA ---\n");

}

bool checkEnd(unsigned char* endMessage, off_t endMessageSize, unsigned char*
startMessage, off_t startMessageSize) {
  if (startMessageSize != endMessageSize)
    return FALSE;
  else {
    if (endMessage[0] == C_END) {
      for (int i = 1; i < startMessageSize; i++)
      {
        if (startMessage[i] != endMessage[i])
          return FALSE;
      }
      return TRUE;
    }
    else
      return FALSE;
  }
  return true;
}

struct readReturn llread(int fd, int* tNumber) {

  struct receiveTramaReturn receiveRet;
  while(true) {
    receiveRet = receiveTrama(tNumber, fd);
    if(receiveRet.statusCode == 0) {
      printf("\nReceived Trama with success!\n \nSendign RR (%d)\n", !(*tNumber));
      sendSupervisionTrama(fd, getCField("RR", !(*tNumber)), A_C_SET);
      break;
    }
    else {
      printf("Didn't receive Trama with success!\n");
      if(receiveRet.statusCode == 2) {
        printf("\n-- Repeated Byte --\n\nSendign RR (%d)\n", !(*tNumber));
        sendSupervisionTrama(fd, getCField("RR",!(*tNumber)), A_C_SET);
      }
      else if(receiveRet.statusCode == -1) { // Send REJ
        printf("\n-- Retransmit Byte --\n\nSendign REJ (%d)\n", *tNumber);
```

```
      sendSupervisionTrama(fd, getCField("REJ", *tNumber), A_C_SET);
    }
   }
  }

  struct readReturn llreadRet;
  llreadRet.currentMessage = receiveRet.currentMessage;
  llreadRet.currentMessageSize = receiveRet.currentMessageSize - 1; // Exclude BCC2

  return llreadRet;
}

off_t sizeOfFile_Start(unsigned char *start)
{
  return (start[3] << 24) | (start[4] << 16) | (start[5] << 8) | (start[6]);
}

unsigned char *nameOfFile_Start(unsigned char *start)
{

  int L2 = (int)start[8];
  unsigned char *name = (unsigned char *)malloc(L2 + 1);

  int i;
  for (i = 0; i < L2; i++)
    name[i] = start[9 + i];

  name[L2] = '\0';
  return name;
}

void createFile(unsigned char *data, off_t* sizeFile, unsigned char filename[])
{
  FILE *file = fopen((char *)filename, "wb+");
  fwrite((void *)data, 1, *sizeFile, file);
  printf("%ld\n", *sizeFile);
  printf("New file created\n");
  fclose(file);
}

void llclose(int fd) {
  printf("\n-- RECEIVED DISC --\n");
  receiveSupervisionTrama(false, getCField("DISC", true), fd, A_C_SET);
  printf("\n-- SENT DISC --\n");
  sendSupervisionTrama(fd, getCField("DISC", true), Other_A);
  receiveSupervisionTrama(false, getCField("UA", true), fd, Other_A);
  printf("\n-- RECEIVED UA --\n");
}

int main(int argc, char** argv)
{
  // srand(time(NULL));
  int fd;
  struct termios oldtio,newtio;
  int tNumber = -1; // [Nr = 0 | 1]
```

```
 if ( (argc < 2) ||
      ((strcmp("/dev/ttyS0", argv[1])!=0) &&
      (strcmp("/dev/ttyS11", 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); }

 llopen(fd, &oldtio, &newtio);

 // Receive start Trama -> Capture fileName and fileTotalSize
 struct readReturn startRet;

 startRet = llread(fd, &tNumber); // Only 1 Frame
 unsigned char* startMessage = startRet.currentMessage;
 off_t startMessageSize = startRet.currentMessageSize;

 printf("\n-- RECEIVED START --\n");

 // Parse it's info
 off_t dataSize = sizeOfFile_Start(startMessage); // File Size

 unsigned char* fileName = nameOfFile_Start(startMessage); // File Name

 printf("-- File Size: %ld --\n", dataSize);
 printf("-- File Name: %s --\n", fileName);

 unsigned char* totalMessage = (unsigned char*) malloc(0);
 off_t totalMessageSize = 0;

 struct readReturn messageRet;
 double percentageLoaded = 0;

 while(true) {
   messageRet = llread(fd, &tNumber);
   if(checkEnd(messageRet.currentMessage, messageRet.currentMessageSize, startMessage,
startMessageSize)) {
     printf("\n-- RECEIVED END --\n");
     break;
   }

   totalMessage = realloc(totalMessage, totalMessageSize + messageRet.currentMessageSize -
DATA_HEADER_LEN);
   memcpy(&totalMessage[totalMessageSize],
&messageRet.currentMessage[DATA_HEADER_LEN], messageRet.currentMessageSize -
DATA_HEADER_LEN);
   totalMessageSize += (messageRet.currentMessageSize - DATA_HEADER_LEN);

   percentageLoaded = (double) totalMessageSize / dataSize;
   printf("\n -- | Percentage Loaded: %f | --\n", percentageLoaded * 100);
   free(messageRet.currentMessage);
 }

 free(startMessage);
```

```
  createFile(totalMessage, &dataSize, fileName);
  free(totalMessage);

  llclose(fd);

  printf("END!\n");

  sleep(1);

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

GeneralFuntions.c:

```
#include "headers/generalFunctions.h"

// C Campo de Controlo
// SET (set up)                    0 0 0 0 0 0 1 1
// DISC (disconnect)                 0 0 0 0 1 0 1 1
// UA (unnumbered acknowledgment)      0 0 0 0 0 1 1 1
// RR (receiver ready / positive ACK)   R 0 0 0 0 1 0 1
// REJ (reject / negative ACK)         R 0 0 0 0 0 0 1     R = N(r)

unsigned char getCField(char typeMessage[25], bool nTrama) {
    if(strcmp(typeMessage, "SET") == 0)
        return C_SET;
    else if(strcmp(typeMessage, "DISC") == 0)
        return C_DISC;
    else if(strcmp(typeMessage, "UA") == 0)
        return C_UA;
    else if(strcmp(typeMessage, "RR") == 0)
        return C_RR(nTrama);
    else // if(strcmp(typeMessage, "REJ") == 0)
        return C_REJ(nTrama);
}

unsigned char computeBcc2(unsigned char* data, int nBytes, int startPosition) {
  int result = data[startPosition];

  for(int i = startPosition + 1; i < startPosition + nBytes; i++)
    result ^= data[i];

  return result;
```

```
}

void configureSerialPort(int fd, struct termios* oldtio, struct termios* newtio) {

  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 = 0;

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

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

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

  tcflush(fd, TCIOFLUSH);

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

int receiveSupervisionTrama(bool withTimeout, unsigned char cField, int fd, unsigned char
aField) {
  if(withTimeout)
    alarm(TIMEOUT);

  unsigned char byte;
  char state[6][25] = { "START", "FLAG_RCV", "A_RCV", "C_RCV", "BCC_OK", "STOP" };
  int i = 0, res;

  // returnState:
  // 1 | readSuccessful
  // 2 | REJ

  int returnState = 1;

  while (strcmp(state[i], "STOP") != 0) {      /* loop for input */

    res = read(fd, &byte, 1);   /* returns after 1 chars have been input */

    if(withTimeout) {
      if(res < 0)  // Read interrupted by a signal
        continue; // Jumps to another iteration
    }
```

```
   if(byte == (aField ^ cField)) { // BCC1
    if(strcmp(state[i], "C_RCV") == 0) {
       i++;
       continue;
    }
   }
   if(byte == cField || byte == C_REJ(0) || byte == C_REJ(1)) { // C
    if(strcmp(state[i], "A_RCV") == 0) {
     if(byte == C_REJ(0)) {
       returnState = 2;
       cField = C_REJ(0);
     }
     else if(byte == C_REJ(1)) {
       returnState = 2;
       cField = C_REJ(1);
     }
     i++;
     continue;
    }
   }
   if(byte == aField) { // A
    if(strcmp(state[i], "FLAG_RCV") == 0) {
     i++;
     continue;
    }
   }

   switch (byte)
   {
    case FLAG_SET:
     if(strcmp(state[i], "START") == 0 || strcmp(state[i], "BCC_OK") == 0)
       i++;
     else
       i = 1; // STATE = FLAG_RCV
     break;
    default: // Other_RCV
     i = 0; // STATE = START
   }
  }

  return returnState;
}

void sendSupervisionTrama(int fd, unsigned char cField, unsigned char aField) {
   unsigned char buf[5];
   buf[0] = FLAG_SET;
   buf[1] = aField;
   buf[2] = cField;
   buf[3] = aField ^ cField;
   buf[4] = FLAG_SET;

   write(fd, buf, sizeof(buf));
}

void checkMaxBytesToSend(off_t* packageSize) {
```

```
 if(N_BYTES_TO_SEND < *packageSize) {
   printf("\n\nN_BYTES_TO_SEND (Data Frame size) has to be greater than the Control Frame
size, which is = %ld\n\n", *packageSize);
   exit(1);
 }
 else if(N_BYTES_TO_SEND > MAX_NUM_OCTETS) {
   printf("%d\n", MAX_NUM_OCTETS);
   printf("\n\nN_BYTES_TO_SEND can\'t be larger than %d. Please choose a smaller value.\n\n",
MAX_NUM_OCTETS);
   exit(1);
 }
}
```

Anexo II

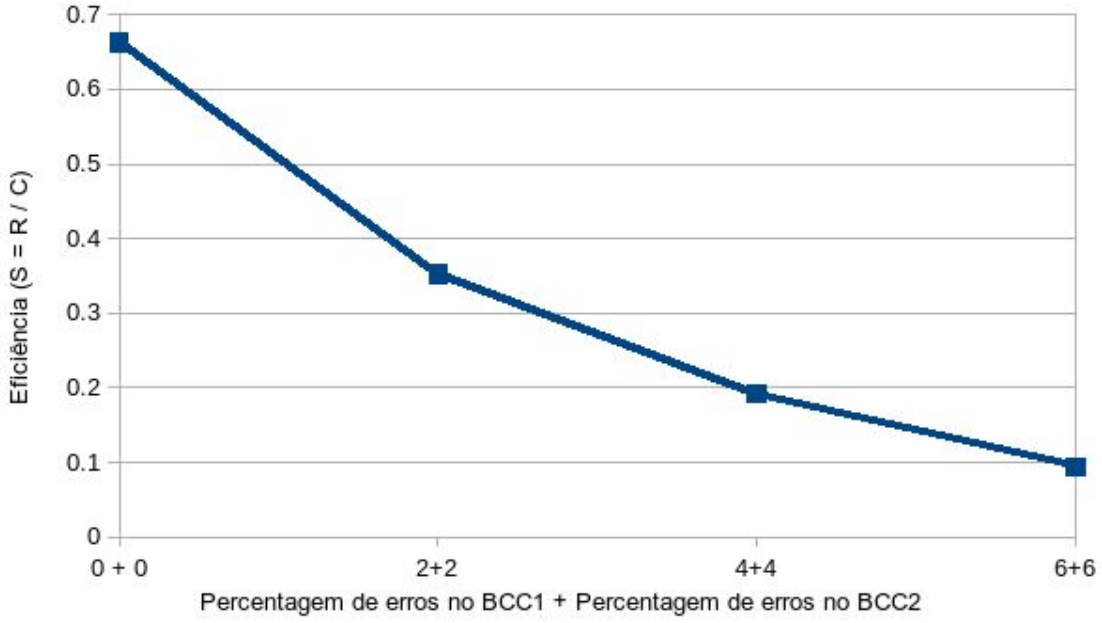

Figura 1. Variação do FER

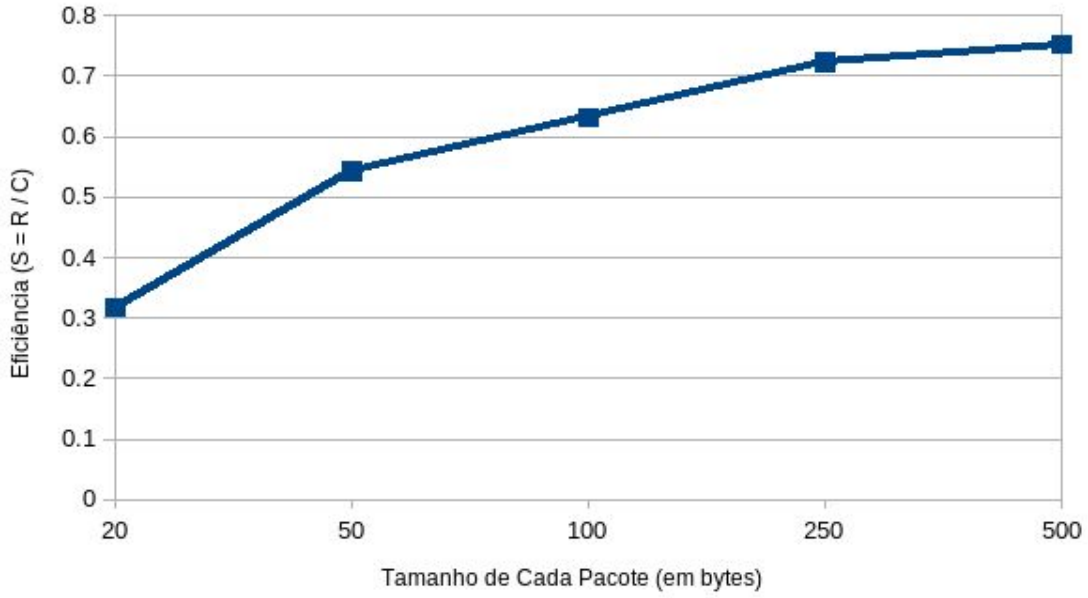

Figura 2. Variação do Tamanho de Cada Pacote (em bytes).

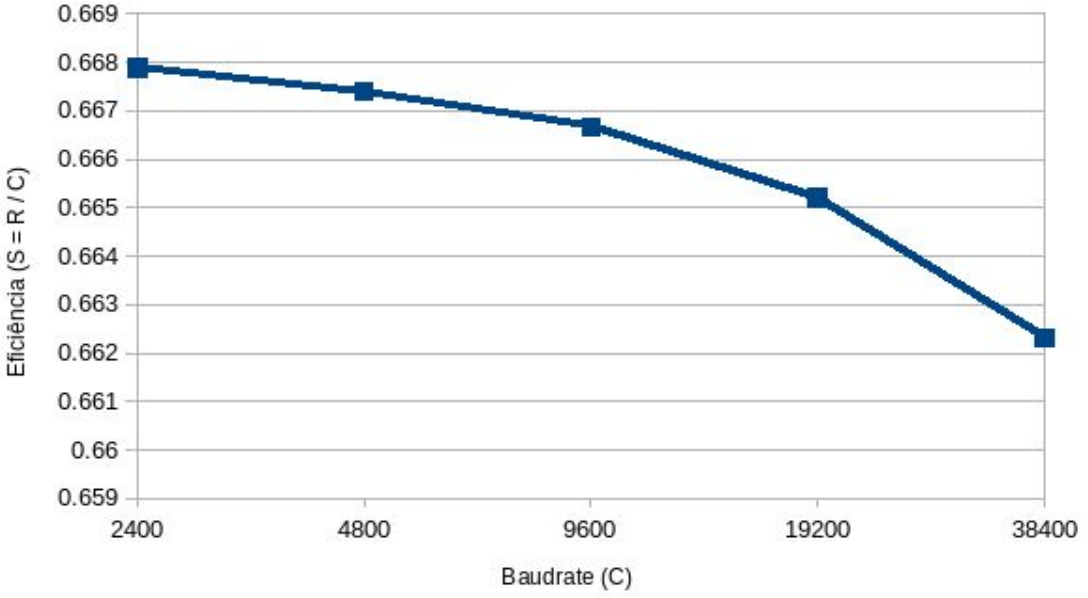

Figura 3. Variação da capacidade de ligação - C.

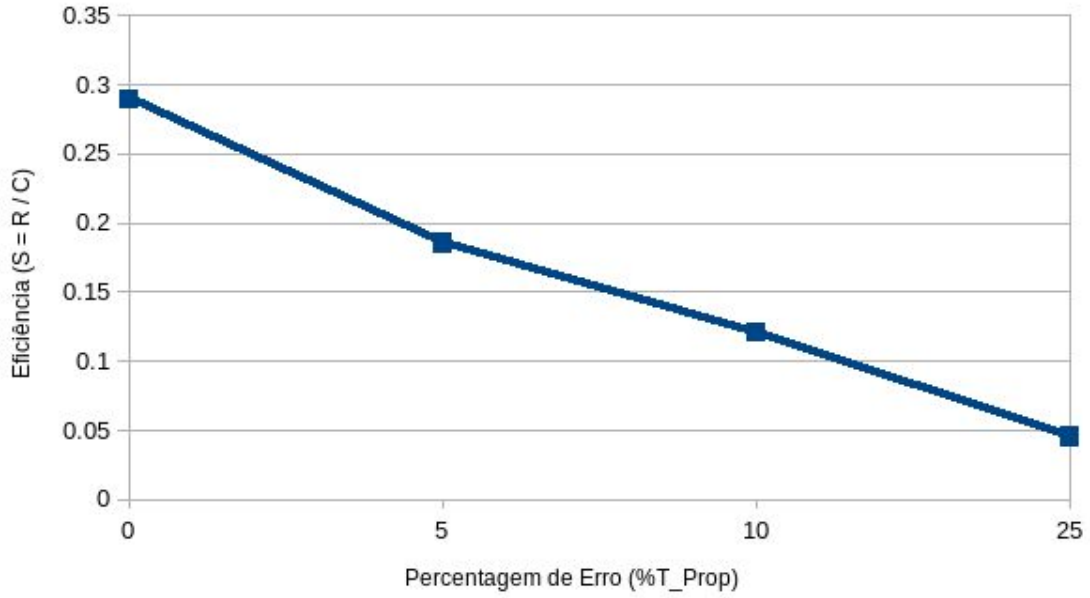

Figura 4. Variação do Tempo de Propagação.

Nº total de bits	87744
Nº total de bytes	10968
C (baudrate)	38400
Tamanho de cada pacote (bytes)	128

Probabilidade de erro (%bcc1 + %bcc2)	Tempo (s)	R (bits/tempo)	S (R/C)	S (R/C) Média
0 + 0	3.447913	25448.4379391	0.66272	0.6627611642052
	3.447482	25451.6194718	0.662803	
2+2	6.524625	13448.1292028	0.350212	0.3523188476966
	6.447044	13609.9583003	0.354426	
4+4	9.638551	9103.44303827	0.237069	0.1920729654002
	15.536069	5647.76070446	0.147077	
6+6	21.641155	4054.49709131	0.105586	0.0941105316246
	27.651654	3173.19173746	0.082635	

Figura 5. Cálculos da eficiência variando a percentagem de erros no BCC1 e no BCC2.

Nº total de bits	87744
Nº total de bytes	10968
C (baudrate)	38400

Tamanho de cada pacote (bytes)	Tempo (s)	R (bits/tempo)	S (R/C)	S (R/C) Média
20	7.189235	12204.9147093	0.317836	0.3174185537188
	7.208184	12172.8302163	0.317001	
50	4.209936	20842.1220655	0.542764	0.5428429071522
	4.208706	20848.2132038	0.542922	
100	3.618299	24250.068886	0.631512	0.6315646664207
	3.617698	24254.0974951	0.631617	
250	3.160965	27758.6116898	0.722881	0.7228911471443
	3.160872	27759.4284109	0.722902	
500	3.03812	28881.0185246	0.75211	0.7520570112802
	3.038547	28876.9599417	0.752004	

Figura 6. Cálculos da eficiência variandoo tamanho da trama I.

Nº total de bits	87744			
Nº total de bytes	10968			

C (baudrate)	Tempo (s)	R (bits/tempo)	S (R/C)	S (R/C) Média
2400	54.74	1602.92290829	0.667885	0.6678845451224
	54.74	1602.92290829	0.667885	
4800	27.39	3203.50492881	0.667397	0.6673968601679
	27.39	3203.50492881	0.667397	
9600	13.71	6400	0.666667	0.6666666666667
	13.71	6400	0.666667	
19200	6.87	12772.0524017	0.665211	0.665211062591
	6.87	12772.0524017	0.665211	
38400	3.45	25433.0434783	0.662319	0.6623188405797
	3.45	25433.0434783	0.662319	

Figura 7. Cálculos da eficiência variando a capacidade de ligação.

Nº total de bits	87744			
Nº total de bytes	10968			
C (baudrate)	38400			

Percentagem de erro (%T_Prop)	Tempo (s)	R (bits/tempo)	S (R/C)	S (R/C) Média
0	3.447719	11137.7986431	0.290047	0.2900781417259
	3.446975	11140.2026414	0.290109	
5	5.3711311	7149.33210251	0.186181	0.1861902146056
	5.370572	7150.0763792	0.1862	
10	8.255442	4651.47717106	0.121132	0.1211385940922
	8.254573	4651.96685522	0.121145	
25	21.718558	1768.07318423	0.046044	0.046041873443
	21.720161	1767.94269619	0.04604	

Figura 8. Cálculos da eficiência variando a percentagem de erros no Tempo de Propagação.