

# Redes de Computadores

## 1º Trabalho Laboratorial – Ligação de Dados

Turma 7

João Praça - up201704748

Leonor Sousa - up201705377

Sílvia Rocha - up201704684

Prof. Manuel Pereira Ricardo

Prof. Rui Lopes Campos

## Sumário

O primeiro projeto laboratorial foi elaborado no contexto da unidade curricular Redes de Computadores e teve como tema ligação de dados. Deste modo o objetivo do projeto passava em elaborar um programa que simulava um protocolo de ligação de dados e um protocolo de uma aplicação que tinham como objetivo a transferência de dados entre dois computadores via porta de série.

## Introdução

Este projeto tinha como **objetivos**:

- Elaborar um protocolo de ligação de dados;
- Elaborar uma aplicação simples que permite a transferência de ficheiros, utilizando o protocolo de ligação de dados previamente elaborado.

O relatório adjacente ao projeto tem como principal objetivo esclarecer a arquitetura, estrutura e a aplicação dos protocolos utilizados no projeto, assim como fazer a validação e a análise da eficiência do código desenvolvido.

Seguidamente, apresentam-se, neste relatório, informações sobre a implementação, casos de uso e resultados deste projeto. Deste modo, o relatório encontra-se estruturada da seguinte forma:

- **Arquitetura e Estrutura do Código** – descrição da arquitetura utilizada e da estruturação do código (APIs, estruturas de dados utilizadas e funções), assim como a relação entre ambas.
- **Casos de Uso Principais** – identificação dos casos de uso mais importantes, assim como da sequência de chamada de funções relacionada com cada um.
- **Protocolos** - descrição dos principais aspetos funcionais e implementação respetiva dos seguintes protocolos:
  - Protocolo de Ligação de Dados
  - Protocolo de Aplicação
- **Validação e Eficiência do Protocolo de Ligação de Dados** – descrição dos testes utilizados e seus respetivos resultados, caracterização estatística da eficiência do protocolo e comparação da mesma com os resultados esperados.

## Arquitetura e Estrutura do Código

A arquitetura deste projeto consiste abrange 3 blocos funcionais:

### **protocol** (protocol.h e protocol.c)

Bloco responsável por fazer a criação e processamento de tramas e por fazer a transmissão em si. É também neste bloco que são efetuados os mecanismos de stuffing e destuffing, assim como eventuais retransmissões e verificação da integridade das tramas.

A API deste bloco é constituída pelas seguintes principais funções:

|   |   |
|---|---|
| int open_port(char* porta, struct termios *oldtio)  | void close_port(int fd, struct termios *oldtio) |
| void send_set(int fd);  | void send_ua_rcv(int fd);                       |
| void send_ua_snd(int fd);   | void send_disc_rcv(int fd);                     |
| void send_disc_snd(int fd);   | void send_resp(int fd, char c, char a);         |
| int send_msg(int fd, unsigned char* msg, int length);   |   |
| int receive_msg(int fd, unsigned char c, unsigned char a, bool data, unsigned char data_buf[], bool data_resp); |   |
| void receive_set(int fd);   | void receive_disc_rcv(int fd);                  |
| void receive_disc_snd(int fd);  | void receive_ua_rcv(int fd);                    |
| void receive_ua_snd(int fd);  | bool receive_data_rsp(int fd);                  |
| int receive_data(int fd, unsigned char data_buf[]);   |   |

### **application** (application.h e application.c)

Bloco responsável por fazer a divisão do ficheiro em diversas partes e por construir e processar os pacotes (de controlo e de dados).

A API deste bloco é constituída pelas seguintes principais funções:

|  |                         |
|--|-------------------------|
| int llopen(unsigned char *porta, bool transmitter);      | int llclose(int fd);    |
| int llwrite(int fd, unsigned char * buffer, int length); | int llreadFile(int fd); |
| void llopen_image(unsigned char *path, int fd);          |                         |

### **nserial** (nserial.c)

Bloco responsável por fazer a interface entre o utilizador e o bloco application. É constituído pela função main e pode ser chamado, via consola, através da instrução:

.\nserial <portaDeSerie> <TRANSMITTER(1)|RECEIVER(0)> <ficheiro>

- <portaDeSerie> é uma string do tipo "/dev/ttySX", em que X é o número da porta de série
- O campo <TRANSMITTER(1)|RECEIVER(0)> deverá ser 1 se se tratar do transmissor e 0 se se tratar do receptor
- O campo <ficheiro> só deverá existir caso se trate do transmissor e representa o ficheiro que deve ser enviado.

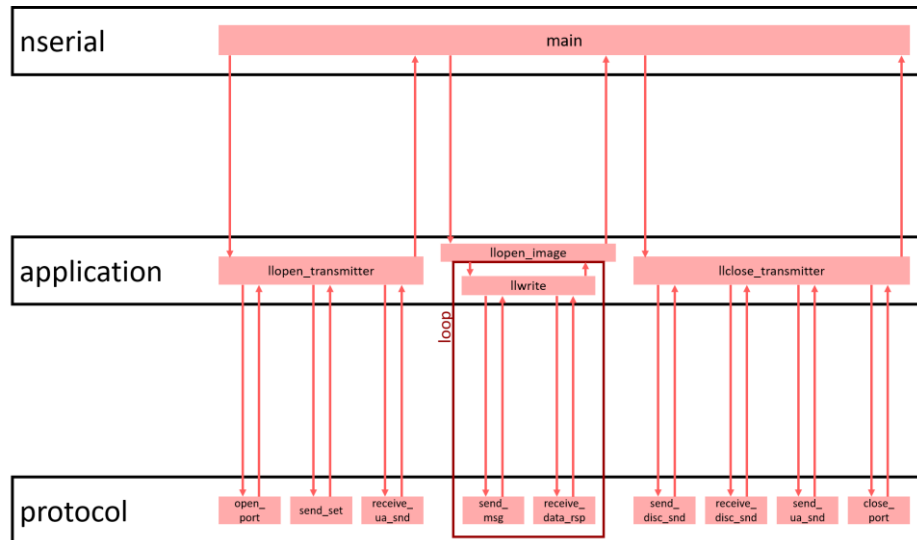
## Casos de Uso Principais

O principal caso de uso deste programa é a transmissão de ficheiros via porta de série. Desta forma, podemos subdividir este caso em dois subcasos:

### Envio de um Ficheiro

O envio de um ficheiro pode ser feito, através da linha de comandos, usando a instrução: `./nserial <portaDeSerie> 1 <ficheiro>`.

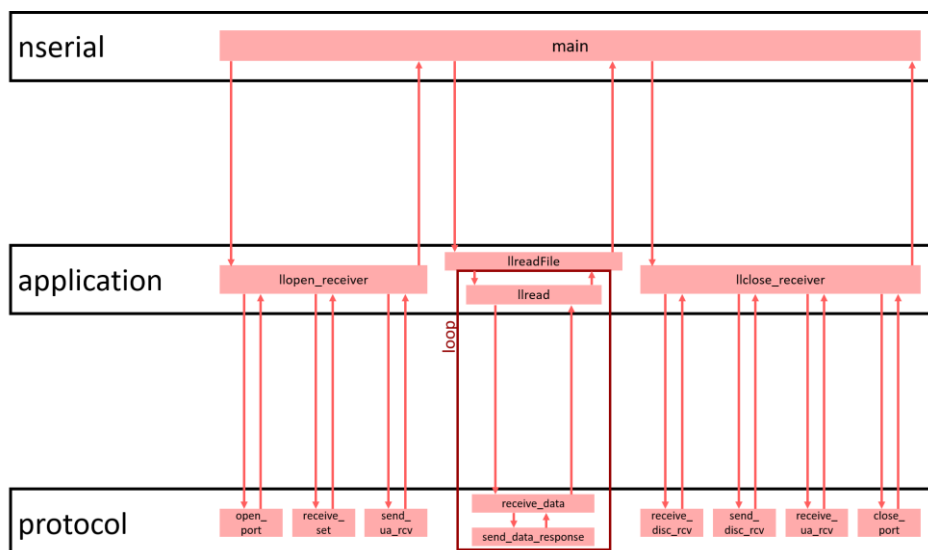
Esta instrução aciona a seguinte sequência de chamada de funções:



### Receção de um Ficheiro

A receção de um ficheiro pode ser feito, através da linha de comandos, usando a instrução: `.\nserial <portaDeSerie> 0`.

Esta instrução aciona a seguinte sequência de chamada de funções:



## Protocolos

### Protocolo de Ligação Lógica

Uma das principais funcionalidades do protocolo de ligação lógico é a criação e o processamento de tramas do tipo I e de tramas de dados. Estas tramas têm um formato já especificado à priori. O seu processamento é efetuado recorrendo à função `receive_msg()`, onde é implementada uma máquina de estados. A sua criação é efetuada nas funções `send_msg()` (tramas de dados) e `send_resp` (tramas do tipo I).

A estas tramas é aplicado byte stuffing (no envio de dados) e byte destuffing (na receção de dados). Este mecanismo é aplicado recorrendo ao octeto escape (0x7D). Desta forma, sempre que se quer enviar o byte 0x7E, são enviados os bytes 0x7D e 0x5E e, sempre que se quer enviar o byte 0x7D, são enviados os bytes 0x7D e 0x5D. Na leitura faz-se o processamento inverso.

O protocolo de ligação de dados faz também a verificação da integridade das tramas, a partir da análise do BCC e do BCC2, no caso de tramas de dados.

No caso de ocorrer um timeout na receção da resposta ao envio da trama, a trama é reenviada. Este mecanismo é efetuado recorrendo a sinais do tipo ALARM.

|   |  |   |  |
|---|--|---|--|
| <pre>int cnt=0; for (int i = 0; i &lt; length; i++) {     if (msg[i] == 0x7E)     {         buf2[4+i+cnt]=0x7D;         cnt++;         buf2[4 + i+cnt] = 0x5E;     }     else if (msg[i] == 0x7D)     {         buf2[4 + i + cnt] = 0x7D;         cnt++;         buf2[4 + i + cnt] = 0x5D;     }     else         buf2[4 + i + cnt] = msg[i]; }</pre> | <pre>if (escape) {     if (msg == ESC1)     {         data_buf[cnt]=0x7E;         cnt++;     }     else if (msg == ESC2)     {         data_buf[cnt]=0x7D;         cnt++;     }     else         state = START; //ERRO     escape = false;     break; } if (msg == ESC){     escape = true;     break; } char msg_string[255]; sprintf(msg_string, "%c", msg); data_buf[cnt]=msg_string[0]; cnt++;</pre> | <pre>char bcc_rcv = data_buf[cnt]; char bcc_real = data_buf[0]; for (int i = 1; i &lt; cnt; i++)     bcc_real = bcc_real ^ data_buf[i]; if (bcc_real == bcc_rcv) {     previous_s = even_bit;     send_data_response(fd, false, false); } else {     even_bit = !even_bit;     send_data_response(fd, true, false); }</pre> | <pre>void alarmSet() {     if (ua_received)         return;     if (num_retr_set &lt; MAX_RETR)     {         send_set(fdG);         num_retr_set++;     }     else     {         exit(1);     } }</pre> |
| 3. Stuffing   | 2. Destuffing  | 4. Verificação da Integridade   | 1. Retransmissão   |

### Protocolo de Aplicação

O protocolo de aplicação é responsável por fazer a construção e o processamento dos pacotes (de dados e de controlo), não tendo qualquer conhecimento da estrutura das tramas e de todos os outros pormenores que fazem parte do protocolo de ligação de lógica. Na construção e desconstrução dos pacotes deve ser efetuada uma numeração correta dos pacotes.

## Validação

Para garantir a validade do protocolo desenvolvido foram efetuados os seguintes testes:

- Envio de ficheiros de vários tamanhos e tipo.
  - Todos os ficheiros
- Interrupção da ligação múltiplas vezes durante o envio de um ficheiro.
  - Em cada uma das medições, do lado do *sender*, o alarme instalado para o envio de dados foi acionado e os dados foram reenviados.
- Geração de interferência durante o envio de um ficheiro.
  - Do lado do *receiver*, os dados incorretos foram descartados .
  - Do lado do *sender*, os dados foram reenviados.
- Variação na percentagem de erros simulados no envio de um ficheiro.
- Variação do tamanho de pacotes de dados no envio de um ficheiro.
- Variação da capacidade de ligação (*baudrate*) no envio de um ficheiro.

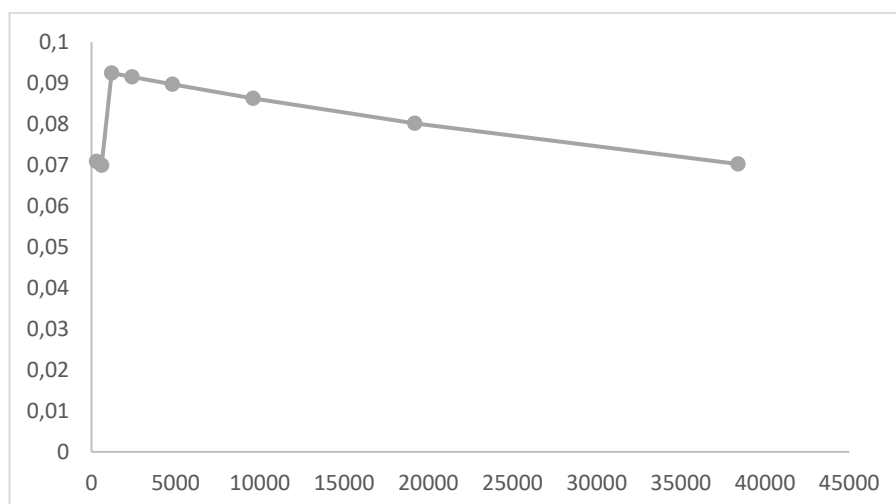
Todos estes testes foram bem sucedidos. Os três últimos testes serão abordados com maior pormenor na secção seguinte.

## Eficiência do protocolo de ligação de dados

Todas as medições de eficiência apresentadas nesta secção foram efetuadas usando como ficheiro de envio o que nos foi fornecido (pinguim.gif) e recorrendo a duas máquinas virtuais. Os valores obtidos em todas as medições (e o cálculo teórico correspondente) e que originaram os gráficos que se seguem encontram-se na secção correspondente aos anexos.

### Variação da Capacidade de Ligação (*baudrate*)

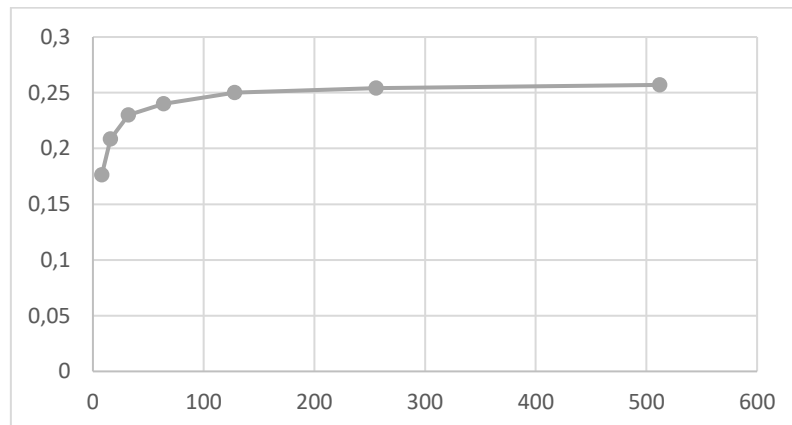
Ao testar o comportamento do protocolo aquando da variação da capacidade de ligação, as alterações do tempo de execução foram bastante visíveis. Observando os valores objetivos inferimos que de facto o aumento da *baudrate* leva a uma diminuição da eficiência. Esta conclusão comprovou-se nos cálculos teóricos da eficiência nas mesmas situações que revelam exatamente a mesma tendência.



### Variação do tamanho dos Pacotes de Dados

Ao testar o comportamento do protocolo aquando da variação do tamanho dos pacotes de dados, verifica-se que a eficiência do mesmo aumenta com o crescimento dos pacotes. Esta

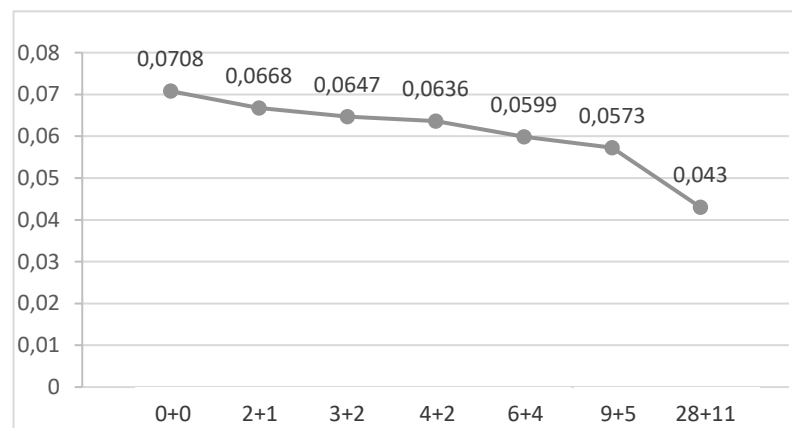
inferência não se verifica no cálculo teórico da eficiência em que os valores diminuem com o aumento dos pacotes.



### Variação do FER

Ao testar o comportamento do protocolo aquando da variação da probabilidade de erros simulados e, consequentemente, o número de erros gerados, é possível concluir que os erros quer ao nível do cabeçalho quer ao nível de dados afetam a eficiência do protocolo. É ainda visível que os erros associados ao bcc1 têm um impacto maior que o bcc2. Este facto é facilmente explicável pelo facto de que aquando de um erro no bcc1, o *sender* irá recorrer ao alarm instalado pelo que será necessário atingir o TIMEOUT para que volte a enviar a mesma trama. Já no bcc2, aquando de um erro, as informações são simplesmente descartadas e a nova trama é enviada imediatamente.

No gráfico seguinte <num>+<num> representa o número de erros no bcc1 e bcc2, respetivamente.



Este protocolo recorre ao mecanismo *Stop & Wait* para controlo de erros. Este mecanismo, permite a retransmissão dos dados em caso de erro.

Após a transmissão de um *packet* de informação, o emissor espera por uma confirmação positiva por parte do recetor, denominada por *acknowledgment*, **ACK**. Quando o recetor recebe o *packet*, caso não tenha nenhum erro, confirma com **ACK**, caso tenha erro, envia **NACK**. Assim que o emissor recebe a resposta do recetor, no caso de **ACK**, continua e envia um novo *packet*, mas no caso de **NACK**, volta a enviar o mesmo *packet*.

Na nossa aplicação, quando o emissor manda qualquer tipo de tramas (U, S ou I) espera uma resposta. Essa resposta é **RR** caso o recetor receba os dados sem erros, e **REJ** caso contrário. Assim, o emissor sabe se deve mandar uma nova trama ou reenviar a mesma. Para que seja possível identificar se uma trama é nova ou duplicada, varia-se o *Nr* destas tramas de resposta de acordo com a trama de *Ns* 0 ou 1 enviada pelo emissor.

## Conclusões

Consideramos que os objetivos principais do projeto foram alcançados com sucesso: o programa lê corretamente um ficheiro, divide-o em pacotes numerados, cria tramas, envia-as através da porta de série (transmissão série assíncrona), recebe-as, escreve o ficheiro no computador recetor, e deteta e corrige eventuais erros nas tramas transmitidas. O código está organizado em camadas independentes (*nserial*, *application* e *protocol*), de forma a que haja abstração.

Para além disso, este projeto ajudou-nos a aprofundar os conhecimentos abordados nas aulas teóricas, nomeadamente a transmissão assíncrona de dados e a sua eficiência, a consolidar conhecimentos em C e na coordenação enquanto grupo.



# Anexo I

nserial.c

```
#include "application.h"

int main(int argc, char **argv)
{
    int fd;
    struct termios oldtio;

    if ((argc != 3 && argc != 4) ||
        ((strcmp("/dev/ttyS0", argv[1]) != 0) &&
         (strcmp("/dev/ttyS1", argv[1]) != 0) &&
         (strcmp("/dev/ttyS2", argv[1]) != 0)))
    {
        printf("Usage:\tnserial SerialPort TRANSMITTER(1)|RECEIVER(0)\n\tt
ex: nserial /dev/ttyS1 1\n");
        exit(1);
    }

    if(atoi(argv[2]) == RECEIVER){
        fd = llopen(argv[1], RECEIVER);

        int size;
        size = llreadFile(fd);

        llclose(fd);
    }
    else if (atoi(argv[2]) == TRANSMITTER){
        fd = llopen(argv[1], TRANSMITTER);

        llopen_image(argv[3], fd);
        llclose(fd);
    }
    else{
        printf("Usage:\tnserial SerialPort TRANSMITTER(1)|RECEIVER(0)\n\tt
ex: nserial /dev/ttyS1 1\n");
        exit(1);
    }
    return 0;
}
```

Application.h

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

```

#include <termios.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>

#define TRANSMITTER 1
#define RECEIVER 0

void send_file();

int llopen(unsigned char *porta, bool transmitter);

int llreadFile(int fd);

int llwrite(int fd, unsigned char * buffer, int length);

int llclose(int fd);

void llopen_image(unsigned char *path, int fd);

```

application.c

```

#include "application.h"
#include "protocol.h"

#define PACKAGE_SIZE 256
struct termios oldtio;
bool trans;

void createDataPackage(unsigned char *package, int indice, int num)
{
    unsigned char* buffer;
    buffer = malloc(sizeof(unsigned char)*(num + 5));
    unsigned char aux[2];
    buffer[0]='1';
    sprintf(aux, "%1x", indice%255);
    buffer[1]=aux[0];
    sprintf(aux, "%d", num/8);
    buffer[2]=aux[0];
    sprintf(aux, "%d", num%8);
    buffer[3]=aux[0];
    for (int i=0; i<num; i++)
        buffer[4+i]=package[i];
    for (int i=0; i<num+4; i++)

```

```

        package[i]=buffer[i];
        free(buffer);
    }

void createControlPackage(unsigned char *buffer, int controlCamp, int fileSize, unsigned char *path)
{
    unsigned char aux[2];
    sprintf(aux, "%d",controlCamp);
    buffer[0]=aux[0];
    buffer[1]='0';
    buffer[2]='1';
    buffer[3]=fileSize;
    buffer[4]='1';
    sprintf(aux, "%d",strlen(path));
    buffer[5]=aux[0];
    buffer[6]='\0';
    strcat(buffer, path);
}

void llopen_image(unsigned char *path, int fd)
{
    FILE* file;
    file = fopen(path, "r");

    if(file==NULL){
        perror("File opening failed ");
        exit(1);
    }
    fseek(file, 0L, SEEK_END);
    int size = ftell(file);
    fseek(file,0L, SEEK_SET);
    unsigned char packageControl[5 + strlen(path)];
    createControlPackage(packageControl, 0x02, size, path);
    llwrite(fd,packageControl, strlen(packageControl));
    int num, i=0;
    do{
        unsigned char *packageBuf;
        packageBuf = malloc(sizeof(unsigned char) * (PACKAGE_SIZE + 4));

        num = fread(packageBuf, sizeof(unsigned char), PACKAGE_SIZE, file);

        packageBuf[num] = '\0';
        createDataPackage(packageBuf, i, num);
        llwrite(fd, packageBuf, num+4);
        i++;
        free(packageBuf);
    } while(num == PACKAGE_SIZE);
    createControlPackage(packageControl,3, size, path);
}

```

```

        llwrite(fd,packageControl, strlen(packageControl));
    }

int llopen_transmitter(unsigned char *porta)
{
    int fd=open_port(porta, &oldtio);
    send_set(fd);
    receive_ua_snd(fd);
    return fd;
}

int llopen_receiver(unsigned char *porta)
{
    int fd=open_port(porta, &oldtio);
    receive_set(fd);
    send_ua_rcv(fd);
    return fd;
}

int llopen(unsigned char *porta, bool transmitter)
{
    trans=transmitter;
    if (transmitter)
        llopen_transmitter(porta);
    else
        llopen_receiver(porta);
}

int llwrite(int fd, unsigned char * buffer, int length){
    int num;
    do{
        num=send_msg(fd, buffer, length);
    } while(!receive_data_rsp(fd));

    return num;
}

int processPackage(unsigned char *buffer, unsigned char *filename, int sequenceNumber)
{
    switch(buffer[0]){
        case '1':{
            unsigned char aux[2];
            sprintf(aux, "%x", sequenceNumber);
            if (buffer[1] == aux[0])
            {
                int k = (int)buffer[2] + (int)buffer[3] * 8;
                unsigned char aux2[PACKAGE_SIZE+4];
                for (int i=0; i < PACKAGE_SIZE+4; i++)

```

```

        aux2[i] = buffer[i];
        for (int i=0; i<k; i++)
            buffer[i] = aux2[i+4];
    }
    return 1;
}

case '2':
    if(buffer[4]=='1'){
        strncpy(filename, &buffer[6], (int)buffer[5]);
    }
    return 2;
case '3':
    return 3;
default:
    return -1;
}
}

int llread(int fd, unsigned char *buffer){
    int num = receive_data(fd, buffer);
    return num;
}

int llreadFile(int fd ){
    unsigned char filename[128];
    unsigned char buf[1024];
    llread(fd, buf);
    processPackage(buf, filename, -1);
    FILE *file;
    filename[0]='z';
    file = fopen(filename, "w");

    unsigned char *aux;
    int num, sequenceNumber = 0;
    int ret;
    do{
        unsigned char buf[1024];
        num = llread(fd, buf);
        ret = processPackage(buf, filename, sequenceNumber);
        if(ret==1){
            fwrite(buf, sizeof(unsigned char), num-4, file);
            sequenceNumber++;
            sequenceNumber %= 255;
        }
    } while(ret != 3);

    fclose(file);
}

```

```

        return num;
    }

    int llclose_transmitter(int fd){
        send_disc_snd(fd);
        receive_disc_snd(fd);
        send_ua_snd(fd);
        close_port(fd, &oldtio);
        return 0;
    }

    int llclose_receiver(int fd){
        receive_disc_rcv(fd);
        send_disc_rcv(fd);
        receive_ua_rcv(fd);
        close_port(fd, &oldtio);
        return 0;
    }

    int llclose(int fd){
        if (trans)
            return llclose_transmitter(fd);
        else
            return llclose_receiver(fd);
    }

```

#### Protocol.h

```

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

int open_port(char* porta, struct termios *oldtio);

void close_port(int fd, struct termios *oldtio);

void send_set(int fd);

void send_ua_rcv(int fd);

```

```

void send_ua_snd(int fd);

void send_disc_rcv(int fd);

void send_disc_snd(int fd);

void send_resp(int fd, char c, char a);

int send_msg(int fd, unsigned char* msg, int length);

int receive_msg(int fd, unsigned char c, unsigned char a, bool data, unsigned char data_buf[], bool data_resp);

void receive_set(int fd);

void receive_disc_rcv(int fd);

void receive_disc_snd(int fd);

void receive_ua_rcv(int fd);

void receive_ua_snd(int fd);

int receive_data(int fd, unsigned char data_buf[]);

bool receive_data_rsp(int fd);

```

protocol.c

```

/*Non-Canonical Input Processing*/

#include "protocol.h"

#define BAUDRATE B38400
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

#define START 0
#define FLAG_RCV 1
#define A_RCV 2
#define C_RCV 3
#define BCC_OK 4
#define RCV_DATA 5
#define BCC2_OK 6
#define STOPS 7

```

```

#define FLAG 0x7E
#define A_RCV_RSP 0x03
#define A_RCV_CMD 0x01
#define A_SND_CMD 0x03
#define A_SND_RSP 0x01
#define SET 0x03
#define UA 0x07
#define DISC 0x0B
#define ESC 0x7D
#define ESC1 0x5E
#define ESC2 0x5D
#define RR_R1 0x85
#define RR_R0 0x05
#define REJ_R1 0x81
#define REJ_R0 0x01
#define C_DATA_S0 0x00
#define C_DATA_S1 0x40

#define MAX_RETR 6
#define TIMEOUT 3

volatile int STOP = FALSE;
bool even_bit = 0;
bool previous_s = 0;
static int num_retr_set = 0;
static int num_retr_disc = 0;
static int num_retr_data = 0;
int fdG;
unsigned char msgG[1024];
int lengthG;
int state;

bool ua_received=false;
bool data_received=false;
bool disc_received=false;

static int erro_cab = 0;
static int erro_dados = 0;

void alarmSet()
{
    if (ua_received)
        return;
    if (num_retr_set < MAX_RETR)
    {
        printf("alarmSet %d \n", num_retr_set);
        send_set(fdG);
        num_retr_set++;
    }
}

```



```

    else
    {
        exit(1);
    }
}

void alarmDisc()
{
    if (disc_received)
        return;
    if (num_retr_disc < MAX_RETR)
    {
        printf("alarmDisc %d \n", num_retr_disc);
        send_disc_snd(fdG);
        num_retr_disc++;
    }
    else
    {
        exit(1);
    }
}

void alarmData()
{
    if (data_received)
        return;
    if (num_retr_data < MAX_RETR)
    {
        printf("alarmData %d \n", num_retr_data);
        send_msg(fdG, msgG, lengthG);
        num_retr_data++;
    }
    else
    {
        exit(1);
    }
}

int open_port(char* porta, struct termios *oldtio)
{
    int fd;
    struct termios newtio;

    /*
        Open serial port device for reading and writing and not as contro
        lling tty
        because we don't want to get killed if linenoise sends CTRL-C.
    */

```

```

fd = open(porta, O_RDWR | O_NOCTTY);
if (fd < 0)
{
    perror(porta);
    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 = 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 1 chars received */

/*
    VTIME e VMIN devem ser alterados de forma a proteger com um tempo
    rizador a
    leitura do(s) pr◊ximo(s) caracter(es)
*/

tcflush(fd, TCIOFLUSH);

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

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

return fd;
}

void close_port(int fd, struct termios *oldtio)
{
    sleep(1);
    tcsetattr(fd, TCSANOW, oldtio);
    close(fd);
}

```

```

    printf("erros cabeçalho: %d\n", erro_cab);
    printf("erros dados: %d\n", erro_dados);
}

void send_resp(int fd, char c, char a)
{
    unsigned char buf2[5];
    buf2[0] = FLAG;
    buf2[1] = a;
    buf2[2] = c;
    buf2[3] = a ^ c;
    buf2[4] = FLAG;
    write(fd, buf2, 5);
}

void send_set(int fd)
{
    if(num_retr_set == 0)
        signal(SIGALRM, alarmSet);
    alarm(TIMEOUT);
    send_resp(fd, SET, A_SND_CMD);
}

void send_ua_rcv(int fd)
{
    send_resp(fd, UA, A_RCV_RSP);
}

void send_ua_snd(int fd)
{
    send_resp(fd, UA, A_SND_RSP);
}

void send_disc_rcv(int fd)
{
    send_resp(fd, DISC, A_RCV_CMD);
}

void send_disc_snd(int fd)
{
    if(num_retr_disc == 0)
        signal(SIGALRM, alarmDisc);
    alarm(TIMEOUT);
    send_resp(fd, DISC, A_SND_CMD);
}

void send_data_response(int fd, bool reject, bool duplicated)
{
    if (reject && duplicated && even_bit)

```

```

        send_resp(fd, RR_R1, A_RCV_RSP);
    else if (reject && duplicated && !even_bit)
        send_resp(fd, RR_R0, A_RCV_RSP);
    else if (reject && even_bit)
        send_resp(fd, REJ_R1, A_RCV_RSP);
    else if (reject && !even_bit)
        send_resp(fd, REJ_R0, A_RCV_RSP);
    else if (!reject && even_bit)
        send_resp(fd, RR_R1, A_RCV_RSP);
    else
        send_resp(fd, RR_R0, A_RCV_RSP);
}

int send_msg(int fd, unsigned char* msg, int length)
{
    if(num_retr_data==0){
        signal(SIGALRM, alarmData);
        fdG=fd;
        for(int i=0; i<length;i++){
            msgG[i]=msg[i];
        }
        lengthG=length;
    }

    alarm(TIMEOUT);
    unsigned char buf2[7 + length*2];
    buf2[0] = FLAG;
    buf2[1] = A_SND_CMD;
    if (even_bit)
        buf2[2] = C_DATA_S0;
    else
        buf2[2] = C_DATA_S1;
    buf2[3] = (A_SND_CMD ^ buf2[2]);

    unsigned char bcc2 = msg[0];
    for (int i = 1; i < length; i++)
        bcc2 = (bcc2 ^ msg[i]);

    int cnt=0;
    for (int i = 0; i < length; i++)
    {
        if (msg[i] == 0x7E)
        {
            buf2[4+i+cnt]=0x7D;
            cnt++;
            buf2[4 + i+cnt] = 0x5E;
        }
        else if (msg[i] == 0x7D)
        {

```

```

        buf2[4 + i + cnt] = 0x7D;
        cnt++;
        buf2[4 + i + cnt] = 0x5D;
    }
    else
        buf2[4 + i + cnt] = msg[i];
}

if (bcc2 == 0x7E)
{
    buf2[4 + length + cnt] = 0x7D;
    cnt++;
    buf2[4 + length + cnt] = 0x5E;
}
else if (bcc2 == 0x7D)
{
    buf2[4 + length + cnt] = 0x7D;
    cnt++;
    buf2[4 + length + cnt] = 0x5D;
}
else
    buf2[4 + length + cnt] = bcc2;

buf2[5 + length + cnt] = FLAG;
buf2[6 + length + cnt] = '\0';
return write(fd, buf2, 6 + length + cnt);
}

int receive_msg(int fd, unsigned char c, unsigned char a, bool data, unsigned char* data_buf, bool data_resp)
{
    state = START;
    int res;
    bool escape = false;
    int cnt = 0;
    int random1, random2;
    while (state != STOPS)
    { /* loop for input */
        unsigned char buf[2];
        res = read(fd, buf, 1); /* returns after 1 char have been input */
        /
        buf[1] = '\0';
        unsigned char msg = buf[0];
        srand(time(0));
        random1 = (rand() % (10 - 1 + 1)) + 1;
        srand(time(0));
        random2 = (rand() % (1000 - 1 + 1)) + 1;
        switch (state)
        {

```

```

case START:
    if (msg == FLAG){
        state = FLAG_RCV;
    }
    break;
case FLAG_RCV:
    if (random1 == 1 && data)
    {
        msg = 0x00;
        a = 0x00;
    }
    if (msg == a)
        state = A_RCV;
    else if (msg != FLAG){
        state = START;
        break;
    }
    break;

case A_RCV:
    if (!data && !data_resp && msg == c)
        state = C_RCV;
    else if (!data && !data_resp)
        state = START;
    else if (data_resp)
    {
        if ((int)msg == RR_R1 || (int)msg == RR_R0 || (int)msg ==
REJ_R1 || (int)msg == REJ_R0)
        {
            data_buf[0] = msg;
            state = C_RCV;
            c=msg;
        }
        else
            state = START;
    }

    else if (msg == C_DATA_S0 || msg == C_DATA_S1)
    {
        if (((bool)msg) != previous_s)
        {
            even_bit = !(bool)msg;
            state = C_RCV;
            c=msg;
        }
        else
        {
            even_bit = (bool)msg;

```

```

        send_data_response(fd, true, (((bool)msg) == previous
_s));

        state = START;
    }
}
else
{
    state=START;
}

break;
case C_RCV:
    if (msg == (a^c)){
        state = BCC_OK;
    }
    else
    {
        state = START;
        erro_cab++;
        send_data_response(fd, true, false);
        return 0;
    }
    break;
case BCC_OK:
    if (random2 == 1 && data)
    {
        msg = 0;
    }
    if (msg == FLAG)
    {
        state = STOPS;
    }
    else if (data)
    {
        state = RCV_DATA;
        if (msg == ESC)
            escape = true;
        else {
            char msg_string[255];
            sprintf(msg_string, "%c", msg);
            data_buf[cnt]=msg_string[0];
            cnt++;
        }
    }
    else{
        state = START;
    }
    break;
case RCV_DATA:

```

```

if (random2 == 1 && data)
{
    msg = 0;
}
if (msg == FLAG)
{
    cnt--;
    char bcc_rcv = data_buf[cnt];
    char bcc_real = data_buf[0];
    for (int i = 1; i < cnt; i++)
        bcc_real = bcc_real ^ data_buf[i];
    if (bcc_real == bcc_rcv)
    {
        previous_s = !even_bit;
        send_data_response(fd, false, false);
    }
    else
    {
        data_buf[0] = '\0';
        even_bit = !even_bit;
        erro_dados++;
        send_data_response(fd, true, false);
    }
    data_buf[cnt] = '\0';
    state = STOPS;
    break;
}
if (escape)
{
    if (msg == ESC1)
    {
        data_buf[cnt]=0x7E;
        cnt++;
    }
    else if (msg == ESC2)
    {
        data_buf[cnt]=0x7D;
        cnt++;
    }
    else
        state = START; //ERRO
    escape = false;
    break;
}
if (msg == ESC){
    escape = true;
    break;
}
char msg_string[255];

```



```

        sprintf(msg_string, "%c", msg);
        data_buf[cnt]=msg_string[0];
        cnt++;
        break;
    default:
        state = START;
        break;
    }
}
return cnt;
}

void receive_set(int fd)
{
    ua_received=false;
    num_retr_set=0;
    receive_msg(fd, SET, A_RCV_RSP, false, NULL, false);
}

void receive_disc_rcv(int fd)
{
    receive_msg(fd, DISC, A_SND_CMD, false, NULL, false);
}

void receive_disc_snd(int fd)
{
    receive_msg(fd, DISC, A_RCV_CMD, false, NULL, false);
    disc_received = true;
}

void receive_ua_rcv(int fd)
{
    receive_msg(fd, UA, A_SND_RSP, false, NULL, false);
}

void receive_ua_snd(int fd)
{
    receive_msg(fd, UA, A_RCV_RSP, false, NULL, false);
    ua_received=true;
}

int receive_data(int fd, unsigned char* data_buf)
{
    int size;
    if (previous_s == 0)
        size= receive_msg(fd, C_DATA_S1, A_SND_CMD, true, data_buf, false
);
    else

```

```

        size = receive_msg(fd, C_DATA_S0, A_SND_CMD, true, data_buf, false);
    };
    return size;
}

bool receive_data_rsp(int fd)
{
    data_received=false;
    num_retr_data=0;
    unsigned char buf[1024];
    receive_msg(fd, RR_R0, A_RCV_RSP, false, buf, true);
    if(buf[0]==REJ_R0 || buf[0] == REJ_R1)
        return false;
    data_received = true;
    even_bit = !even_bit;
    return true;
}

```

## Anexo 2

### Medição tempo de execução e eficiência

| Baudrate | Tempo   |         |         | Tempo Médio | R        | S        |
|----------|---------|---------|---------|-------------|----------|----------|
| 300      | 516,057 | 516,053 | 516,049 | 516,057     | 21,25347 | 0,070845 |
| 600      | 261,442 | 261,440 | 261,444 | 261,442     | 41,95194 | 0,06992  |
| 1200     | 98,867  | 98,867  | 98,867  | 98,867      | 110,9369 | 0,092447 |
| 2400     | 49,934  | 49,936  | 49,934  | 49,934      | 219,6499 | 0,091521 |
| 4800     | 25,474  | 25,474  | 25,474  | 25,474      | 430,5566 | 0,089699 |
| 9600     | 13,244  | 13,244  | 13,244  | 13,244      | 828,1486 | 0,086265 |
| 19200    | 7,128   | 7,128   | 7,127   | 7,128       | 1538,721 | 0,080142 |
| 38400    | 4,066   | 4,067   | 4,067   | 4,067       | 2696,828 | 0,07023  |

| Tamanho | Tempo |       |       | Tempo Médio | R        | S     |
|---------|-------|-------|-------|-------------|----------|-------|
| 8       | 1,675 | 1,625 | 1,564 | 1,621       | 6764,803 | 0,176 |
| 16      | 1,373 | 1,353 | 1,385 | 1,370       | 8003,892 | 0,208 |
| 32      | 1,25  | 1,243 | 1,233 | 1,242       | 8830,918 | 0,230 |
| 64      | 1,182 | 1,179 | 1,209 | 1,190       | 9216,807 | 0,240 |
| 128     | 1,138 | 1,129 | 1,158 | 1,142       | 9607,007 | 0,250 |
| 256     | 1,125 | 1,125 | 1,123 | 1,124       | 9755,114 | 0,254 |
| 512     | 1,116 | 1,116 | 1,102 | 1,111       | 9869,226 | 0,257 |

| Erros BCC1 | Erros BCC2 | Tempo | R        | S        |
|------------|------------|-------|----------|----------|
| 2          | 1          | 4,275 | 2565,614 | 0,066813 |
| 3          | 2          | 4,418 | 2482,571 | 0,06465  |
| 4          | 2          | 4,488 | 2443,85  | 0,063642 |
| 6          | 4          | 4,767 | 2300,818 | 0,059917 |
| 9          | 5          | 4,984 | 2200,642 | 0,057308 |
| 28         | 11         | 6,659 | 1647,094 | 0,042893 |

### Cálculo teórico da eficiência

Cálculos efetuados considerando  $d=4$  e  $V=300 \times 10^6$  (velocidade da luz).

| Tamanho | Tf (L/R) | A (Tprop/Tf) | S (1/(1+2*a)) |
|---------|----------|--------------|---------------|
| 8       | 0,000208 | 0,000064     | 1,000         |
| 16      | 0,000417 | 0,0000375    | 1,000         |
| 32      | 0,000833 | 4,39453E-05  | 1,000         |
| 64      | 0,001667 | 0,000102997  | 1,000         |
| 128     | 0,003333 | 0,000482798  | 0,999         |
| 256     | 0,006667 | 0,004526228  | 0,991         |
| 512     | 0,013333 | 0,08486677   | 0,855         |

$R=38400$  bits/s.

| <i>Baudrate</i> | Tf (L/R) | A (Tprop/Tf) | S (1/(1+2*a)) |
|-----------------|----------|--------------|---------------|
| 300             | 0,853333 | 1,5625E-08   | 1,000         |
| 600             | 0,426667 | 3,66211E-08  | 1,000         |
| 1200            | 0,213333 | 1,71661E-07  | 1,000         |
| 2400            | 0,106667 | 1,60933E-06  | 1,000         |
| 4800            | 0,053333 | 3,01749E-05  | 1,000         |
| 9600            | 0,026667 | 0,001131557  | 0,998         |
| 19200           | 0,013333 | 0,08486677   | 0,855         |
| 38400           | 0,006667 | 12,73001544  | 0,038         |

L=256bits.