

**Redes de Computadores**

Protocolo de Ligação de Dados

Mestrado Integrado em Engenharia Informática e Computação

Gonçalo Teixeira up201806562@fe.up.pt

Gonçalo Alves up201806451@fe.up.pt

Índice

[Sumário 3](#_Toc54518015)

[Introdução 4](#_Toc54518016)

[Arquitetura 5](#_Toc54518017)

[Estrutura de código 6](#_Toc54518018)

[**Writenoncanonical** 6](#_Toc54518019)

[**Noncanonical** 6](#_Toc54518020)

[**Macros** 6](#_Toc54518021)

[**App\_structs** 6](#_Toc54518022)

[**Utils** 7](#_Toc54518023)

[**Data\_link** 7](#_Toc54518024)

[**Files** 7](#_Toc54518025)

[**App\_writer** 8](#_Toc54518026)

[**App\_reader** 8](#_Toc54518027)

[Casos de uso principais 9](#_Toc54518028)

[Protocolo de ligação lógica 10](#_Toc54518029)

[**llopen** 10](#_Toc54518030)

[**llclose** 10](#_Toc54518031)

[**llwrite** 10](#_Toc54518032)

[**llread** 10](#_Toc54518033)

[Protocolo de aplicação 11](#_Toc54518034)

[**generate\_control\_packet** 11](#_Toc54518035)

[**generate\_data\_packet** 11](#_Toc54518036)

[**split\_file** 11](#_Toc54518037)

[**get\_file\_size** 11](#_Toc54518038)

[**read\_file** 11](#_Toc54518039)

[**join\_file** 11](#_Toc54518040)

[**write\_file** 11](#_Toc54518041)

[*Validação* 12](#_Toc54518042)

[*Eficiência do protocolo de ligação de dados* 13](#_Toc54518043)

[**Variação do FER** 13](#_Toc54518044)

[**Variação do tamanho das tramas I** 13](#_Toc54518045)

[**Variação da capacidade da ligação (C)** 13](#_Toc54518046)

[*Conclusões* 14](#_Toc54518047)

[*Anexo I – Código fonte* 15](#_Toc54518048)

[app\_reader.c 15](#_Toc54518049)

[app\_structs.h 16](#_Toc54518050)

[app\_writer.c 17](#_Toc54518051)

[data\_link.c 20](#_Toc54518052)

[data\_link.h 30](#_Toc54518053)

[files.c 31](#_Toc54518054)

[files.h 32](#_Toc54518055)

[macros.h 32](#_Toc54518056)

[noncanonical.c 33](#_Toc54518057)

[noncanonical.h 34](#_Toc54518058)

[utils.c 34](#_Toc54518059)

[utils.h 38](#_Toc54518060)

[writenoncanonical.c 39](#_Toc54518061)

[writenoncanonical.h 40](#_Toc54518062)

# Sumário

Este relatório foi elaborado no âmbito da unidade curricular de Redes e Computadores, e trata-se da implementação de um protocolo de transferência de dados. O trabalho consiste no desenvolvimento de uma aplicação capaz de transferir ficheiros de um computador para o outro através de uma porta série.

O trabalho foi concluído e a aplicação desenvolvida é capaz de transferir ficheiros sem perda de dados.

# Introdução

O objetivo deste trabalho consistiu na implementação de um protocolo de ligação de dados, de acordo com o guião fornecido, e no teste do dito protocolo, através de uma aplicação de transferência de dados. Quanto ao relatório, o seu objetivo é detalhar a componente teórica do trabalho, com a estrutura descrita em baixo:

* **Arquitetura**
  + Descrição dos blocos funcionais e das interfaces implementadas
* **Estrutura do código**
  + Descrição das APIs, principais estruturas de dados e funções e as suas relações com a arquitetura
* **Casos de uso principais**
  + Identificação dos casos de uso
  + Sequências de chamada de funções
* **Protocolo de ligação lógica**
  + Identificação dos principais aspetos funcionais
  + Descrição da estratégia de implementação destes, com a exibição de extratos de código
* **Protocolo de aplicação**
  + Identificação dos principais aspetos funcionais
  + Descrição da estratégia de implementação destes, com a exibição de extratos de código
* **Validação**
  + Descrição dos testes efetuados
  + Apresentação quantificada dos resultados
* **Eficiência do protocolo de ligação de dados**
  + Caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido
* **Conclusão**
  + Síntese da informação apresentada nas secções anteriores
  + Reflexão sobre os objetivos de aprendizagem alcançados.

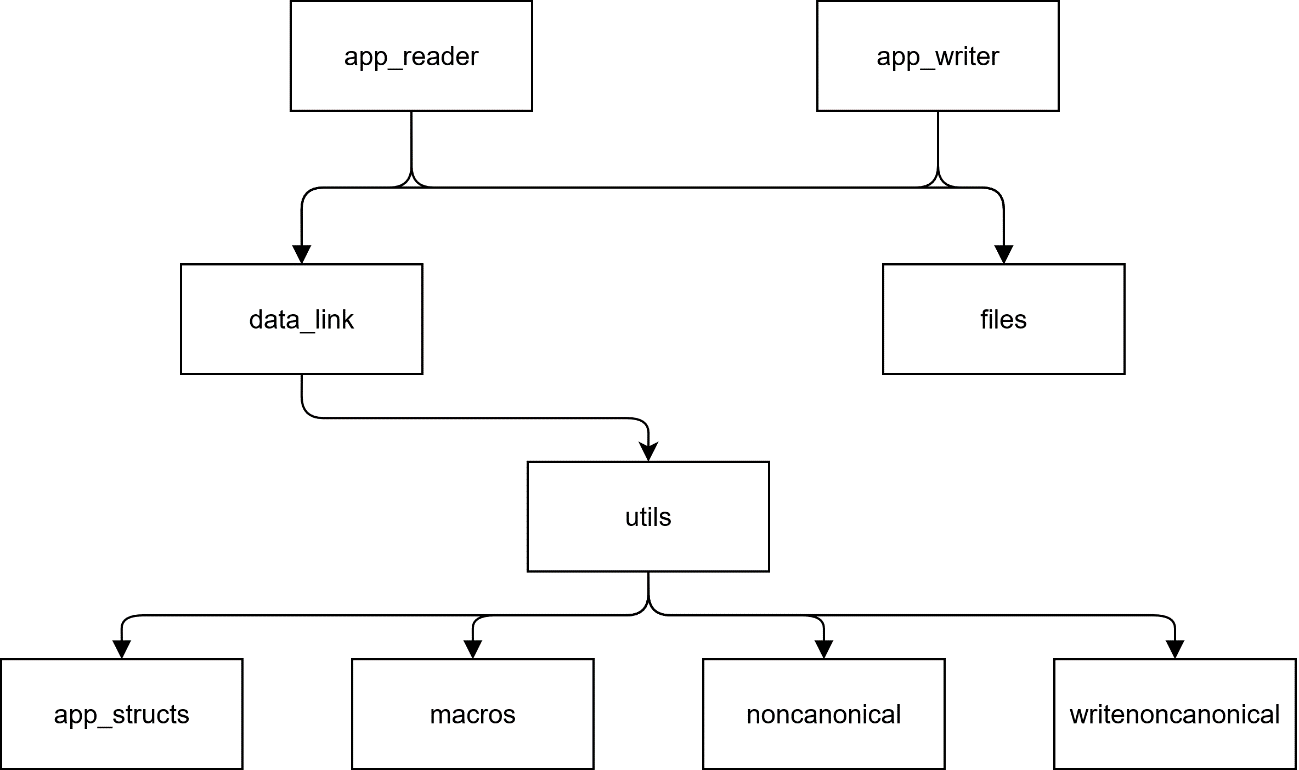
# Arquitetura

O trabalho está dividido em duas camadas: a camada do **de ligação de dados** e a camada da **aplicação**.

A camada de ligação de dados é responsável pelas interações com a porta de série, tal como: a abertura, o fecho, a leitura e a escrita desta. Além disso, a camada também é responsável pelo tratamento das tramas: delineação, *stuffing*, proteção e retransmissão destas.

A camada da aplicação é responsável pelo envio e receção de ficheiros, fazendo uso da interface da camada de ligação de dados. Além disso, a camada também é responsável pelo processamento do serviço: tratamento de cabeçalhos, distinção entre pacotes de controlo e de dados e numeração destes.

# Estrutura de código

O código está dividido em diversos ficheiros de código, tendo uma hierarquia associada a estes, como se demonstra no diagrama seguinte:

Assim, a estrutura do código será feita de forma hierárquica, de baixo para cima.

## **Writenoncanonical**

* **atende** – *Handler* do sinal **SIGALRM**;
* **open\_writer** – Abre a porta série do emissor e implementa o *handler* do alarme;
* **close\_writer** – Fecha a porta série do emissor;

## **Noncanonical**

* **open\_reader** – Abre a porta série do recetor;
* **close\_reader** – Fecha a porta série do recetor;

## **Macros**

O ficheiro macros.h contém, tal como o nome indica, macros importantes para o programa, entre as quais:

* **TRIES** – número de tentativas de escrita;
* **TIMEOUT** – número de segundos de espera por resposta;

## **App\_structs**

O ficheiro app\_structs.h contém as estruturas de dados relevantes para o programa:

* **information\_frame\_t** – *Frame* contendo: *address*, *control*, *bcc*, *data*, *data\_size*, *bcc2* e *raw\_bytes*;
* **data\_packet\_t** – Pacote de dados contendo: *control*, *sequence*, *data\_field\_size*, *data*, *raw\_bytes* e *raw\_bytes\_size*;
* **control\_packet\_t** – Pacote de dados contendo: *control*, *file\_size*, *file\_name*, *filesize\_size*, *raw\_bytes* e *raw\_bytes\_size*;
* **file\_t** – Ficheiro contendo: *data*, *name* e *size*;

## **Utils**

* **parse\_control\_packet** – Função que faz o *parse* de bytes para a estrutura **control\_packet\_t**;
* **parse\_data\_packet** – Função que faz o *parse* de bytes para a estrutura **data\_packet\_t**;
* **print\_control\_packet –** Função que imprime um pacote de controlo;
* **print\_data\_packet –** Função que imprime um pacote de dados. Além disso, permite mostrar todos os bytes de dados, através do uso de uma flag;
* **print\_elapsed\_time –** Função que imprime o tempo passado entre uma escrita e uma leitura;
* **verify\_message –** Função que verifica erros numa trama de Informação, através do bcc;
* **print\_message –** Função que imprime uma trama de Informação;
* **check\_connection–** Função que verifica se a porta série não fechou;
* **array\_to\_number–** Função que transforma um *array* num número de 8 bytes;
* **number\_to\_array–** Função que transforma um número de 8 bytes num *array*;

## **Data\_link**

* **send\_supervision\_frame** – Recebe uma porta e envia-lhe uma mensagem de controlo;
* **receive\_supervision\_frame** – Recebe uma porta e lê uma mensagem de controlo dela;
* **receive\_acknowledgment** – Recebe uma mensagem de **ACK** e retorna o seu *byte* de controlo;
* **send\_acknowledgment** – Envia uma mensagem de **ACK**;
* **receive\_set** – Função que espera receber uma mensagem **SET** e envia **UA** de seguida;
* **send\_set** – Função que envia uma mensagem **SET** e espera receber **UA** de seguida;
* **disconnect\_writer** – Função que envia uma mensagem de **DISC**, espera receber **DISC** de volta e envia **UA**;
* **disconnect\_reader** - Função que recebe uma mensagem de **DISC**, espera enviar **DISC** de volta e recebe **UA**;
* **llopen** – Função que abre a porta série do emissor/recetor e inicia a ligação de dados, retornando o descritor correspondente;
* **llclose** – Função que fecha a porta série do emissor/recetor e termina a ligação de dados;
* **llwrite** – Função que faz o *stuffing* de um pacote de dados e envia para o recetor, esperando receber uma mensagem de **ACK** de volta e procede de acordo com esta;
* **llread** – Função que lê uma mensagem do emissor, faz o *destuffing*, verifica a mensagem e envia uma mensagem **ACK** adequada;

## **Files**

* **get\_file\_size –** Função que calcula o tamanho de um ficheiro, em bytes;
* **read\_file –** Função que lê os dados de um ficheiro;
* **split\_file –** Função que obtém bytes de um ficheiro entre um índice inicial e final, inclusive;
* **join\_file –** Função que concatena pacotes;
* **write\_file –** Função que cria uma cópia do ficheiro recebido;

## **App\_writer**

* **generate\_control\_packet –** Função que cria um pacote de controlo;
* **generate\_data\_packet –** Função que cria um pacote de dados;
* **main –** Função responsável pela escrita de um ficheiro;

## **App\_reader**

* **main –** Função responsável pela leitura de um ficheiro;

# Casos de uso principais

Os casos de uso principais da aplicação são: a interface, que permite a escolha do ficheiro que o emissor pretende enviar, e a transferência do ficheiro, através da porta série.

De modo a dar-se a transferência do ficheiro, o utilizador necessitará de introduzir o número da porta série a ser utilizada, como por exemplo **11**. Adicionalmente, caso se trate do emissor, também terá de inserir o ficheiro a ser enviado, como por exemplo **pinguim.gif**

A transmissão de dados dá-se pela seguinte ordem:

* Abertura da ligação entre os computadores;
* Geração dos pacotes **START** e **STOP**, para controlo;
* Escrita do pacote **START**;
* Escrita dos pacotes de dados;
* Escrita do pacote **STOP**;
* Fecho da ligação entre os computadores;

A receção de dados dá-se pela seguinte ordem:

* Abertura da ligação entre os computadores;
* Leitura e impressão do pacote **START**;
* Leitura e impressão dos pacotes de dados;
* Leitura e impressão do pacote **STOP**;
* Impressão da mensagem completa;
* Fecho da ligação entre os computadores;

# Protocolo de ligação lógica

## **llopen**

Esta função tem a responsabilidade de estabelecer a ligação entre o emissor e o recetor.

No caso do emissor, a porta série é aberta, é enviada uma mensagem **SET** e é esperada uma mensagem **UA**.

No caso do recetor, a porta de série é aberta, é recebida uma mensagem **SET** e é enviada uma mensagem **UA**.

## **llclose**

Esta função tem a responsabilidade de fechar a ligação entre o emissor e o recetor.

No caso do emissor, é enviada uma mensagem **DISC**, posteriormente é recebida uma mensagem igual de volta, é enviada uma mensagem **UA** e a ligação termina.

No caso do recetor, é recebida uma mensagem **DISC**, de seguida é enviada essa mesma mensagem para o recetor, é esperada uma mensagem **UA** e a ligação é terminada.

## **llwrite**

Esta função é responsável pelo envio de tramas.

Inicialmente, é composto o *header* da mensagem: *address*, *control* e *bcc1*. De seguida, é feito o *stuffing* da mensagem e a construção do *bcc2* (também com *stuffing*). Finalmente, a função irá enviar a mensagem completa para o emissor e esperar pela mensagem **ACK**. Dependendo desta, o emissor poderá: continuar a transmissão do ficheiro, passando para o próximo pacote; retransmitir o pacote acabado de enviar, devido a um erro. A retransmissão de um pacote também se pode dar quando o tempo de espera de uma resposta exceder o tempo máximo de espera, **TIMEOUT**.

## **llread**

Esta é função é responsável pela receção de tramas.

Inicialmente, é feita uma leitura da porta série, caractere a caractere. De seguida, é feito o *destuffing* da mensagem e esta é guardada numa estrutura de dados (**information\_frame\_t**). Finalmente, é feita uma verificação de erros e, dependendo do resultado desta, é enviada a mensagem adequada para o emissor.

# Protocolo de aplicação

O protoloco de aplicação implementado tem como aspetos principais:

* Envio de pacotes de controlo **START** e **STOP**. Estes contêm o nome e o tamanho do ficheiro a ser enviado;
* Divisão do ficheiro em pacotes, no emissor, e a concatenação dos pacotes recebidos, no recetor;
* Encapsulamento de cada pacote de dados com um *header* contendo o número de sequência do pacote e o tamanho do pacote;
* Leitura do ficheiro a enviar, no emissor, e criação do ficheiro, no recetor.

Estas funcionalidades foram implementadas usando funções descritas a seguir.

## **generate\_control\_packet**

Esta função retorna um pacote **START** ou **STOP**, recebendo como argumento um inteiro de modo a identificar o tipo de pacote. Estes pacotes serão enviados usando a função **llwrite**, pertencente ao protocolo de ligação de dados.

## **generate\_data\_packet**

Esta função retorna um pacote de dados, recebendo como argumentos: dados de um ficheiro, tamanho dos dados e o número de sequência. Estes pacotes serão enviados usando a função **llwrite**, pertencente ao protocolo de ligação de dados.

## **split\_file**

Esta função retorna dados, recebendo como argumentos: o ficheiro de onde se quer obter os dados, o índice do primeiro byte a recolher e o índice do último byte a receber. Estes dados serão usados na função acima mencionada.

## **get\_file\_size**

Esta função retorna o tamanho de um ficheiro, recebendo como argumento o descritor de um ficheiro.

## **read\_file**

Esta função retorna os dados de um ficheiro, recebendo como como argumentos: o descritor de um ficheiro e o tamanho deste.

## **join\_file**

Esta função recebe como argumentos: um *array* onde se vão concatenar os pacotes, um pacote, o tamanho do pacote e o índice dos dados.

## **write\_file**

Esta função cria uma cópia do ficheiro recebido, recebendo com argumentos: o nome do ficheiro, os *bytes* deste e o seu tamanho.

# *Validação*

Ainda não foram feitos testes

# *Eficiência do protocolo de ligação de dados*

## **Variação do FER**

## **Variação do tamanho das tramas I**

## **Variação da capacidade da ligação (C)**

# *Conclusões*

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

Adicionalmente, foi nos dado a conhecer a **independência entre camadas**, e cada um dos blocos funcionais da arquitetura da aplicação desenvolvida cumpre esta independência.

Na camada de ligação de dados não é feito qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de Informação. Ao nível da ligação de dados não existe qualquer distinção entre pacotes de controlo e de dados, nem é relevante (nem tida em conta) a numeração dos pacotes de dados.

Na camada de aplicação não são conhecidos os detalhes do protocolo de ligação de dados, mas apenas a forma como se acede ao serviço. O protocolo de aplicação desconhece a estrutura das tramas e o respetivo mecanismo de delineação, a existência de *stuffing* (e qual a opção adotada), o mecanismo de proteção das tramas, eventuais retransmissões de tramas de Informação, etc.

Concluindo, o protocolo de ligação de dados foi realizado com sucesso, tendo-se cumprido todos os objetivos, e o desenvolvimento deste contribuiu para um aprofundamento do conhecimento, tanto teórico como prático, deste tema.

# *Anexo I – Código fonte*

## app\_reader.c

#include "data\_link.h"

#include "files.h"

file\_t file;

int main(int argc, char \*argv[]) {

if (argc != 2) {

printf("Usage: %s <number for serial port>\n", argv[0]);

printf("\nExample: %s 11\t-\tfor '/dev/ttyS11'\n", argv[0]);

exit(ERROR);

}

/\* opens transmiter file descriptor on second layer \*/

int receiver\_fd = llopen(atoi(argv[1]), RECEIVER);

/\* in case there's an error oppening the port \*/

if (receiver\_fd == ERROR) {

exit(ERROR);

}

char buffer[1024];

int size;

int state = 0;

// \* START Control Packet

while (state == 0) {

memset(buffer, 0, sizeof(buffer));

while ((size = llread(receiver\_fd, buffer)) == ERROR) {

printf("Error reading\n");

}

control\_packet\_t packet = parse\_control\_packet(buffer, size);

file.size = array\_to\_number(packet.file\_size, packet.filesize\_size);

file.name = packet.file\_name;

print\_control\_packet(packet);

if (packet.control == START) {

state = 1;

}

}

// \* DATA Packets

unsigned char \*full\_message = (unsigned char\*) malloc (file.size);

int index = 0;

while (state == 1) {

memset(buffer, 0, sizeof(buffer));

while ((size = llread(receiver\_fd, buffer)) == ERROR) {

printf("Error reading\n");

}

if (buffer[0] == STOP) {

state = 2;

break;

}

data\_packet\_t data = parse\_data\_packet(buffer, size);

if (data.control != DATA) continue;

print\_data\_packet(&data, FALSE);

join\_file(full\_message, data.data, data.data\_field\_size, index);

index += data.data\_field\_size;

}

// \* STOP Control Packet

if (state == 2) {

control\_packet\_t packet = parse\_control\_packet(buffer, size);

print\_control\_packet(packet);

write\_file("pinguim\_clone.gif", full\_message, file.size);

printf("Received file\n");

}

/\* resets and closes the receiver fd for the port \*/

llclose(receiver\_fd, RECEIVER);

return 0;

}

## app\_structs.h

typedef struct {

unsigned char address;

unsigned char control;

unsigned char bcc1;

unsigned char \*data;

int data\_size; /\* size of the data array \*/

unsigned char bcc2;

unsigned char \*raw\_bytes; /\* full set of bytes for the message \*/

} information\_frame\_t;

typedef struct {

unsigned char control;

unsigned char sequence;

int data\_field\_size;

unsigned char data[1024];

unsigned char \*raw\_bytes;

int raw\_bytes\_size;

} data\_packet\_t;

typedef struct {

unsigned char control;

unsigned char \*file\_size;

unsigned char \*file\_name;

unsigned int filesize\_size;

unsigned char \*raw\_bytes;

int raw\_bytes\_size;

} control\_packet\_t;

typedef struct {

unsigned char\* data;

unsigned char\* name;

unsigned long size;

} file\_t;

## app\_writer.c

#include "data\_link.h"

#include "files.h"

extern int flag;

FILE \*fp;

file\_t file;

control\_packet\_t generate\_control\_packet(int control) {

control\_packet\_t c\_packet;

c\_packet.control = control;

c\_packet.file\_name = file.name;

unsigned char buf[sizeof(unsigned long)];

int num = number\_to\_array(file.size, buf);

c\_packet.file\_size = (unsigned char \*)malloc(num);

memcpy(c\_packet.file\_size, buf, num);

c\_packet.filesize\_size = num;

int i = 0;

// control packet

c\_packet.raw\_bytes = (unsigned char \*)malloc(i + 1);

c\_packet.raw\_bytes[i++] = c\_packet.control;

c\_packet.raw\_bytes = (unsigned char \*)realloc(c\_packet.raw\_bytes, (i + 1));

// file size

c\_packet.raw\_bytes[i++] = FILE\_SIZE;

c\_packet.raw\_bytes = (unsigned char \*)realloc(c\_packet.raw\_bytes, (i + 1));

c\_packet.raw\_bytes[i++] = c\_packet.filesize\_size;

for (int j = 0; j < c\_packet.filesize\_size; j++) {

c\_packet.raw\_bytes = (unsigned char \*)realloc(c\_packet.raw\_bytes, (i + 1));

c\_packet.raw\_bytes[i++] = c\_packet.file\_size[j];

}

c\_packet.raw\_bytes = (unsigned char \*)realloc(c\_packet.raw\_bytes, (i + 1));

// file name

c\_packet.raw\_bytes[i++] = FILE\_NAME;

c\_packet.raw\_bytes = (unsigned char \*)realloc(c\_packet.raw\_bytes, (i + 1));

c\_packet.raw\_bytes[i++] = strlen(c\_packet.file\_name);

c\_packet.raw\_bytes = (unsigned char \*)realloc(c\_packet.raw\_bytes, (i + 1));

for (int j = 0; j < strlen(c\_packet.file\_name); j++) {

c\_packet.raw\_bytes[i++] = c\_packet.file\_name[j];

c\_packet.raw\_bytes = (unsigned char \*)realloc(c\_packet.raw\_bytes, (i + 1));

}

c\_packet.raw\_bytes\_size = i;

return c\_packet;

}

data\_packet\_t generate\_data\_packet(unsigned char \*buffer, int size, int sequence) {

data\_packet\_t d\_packet;

d\_packet.control = DATA;

d\_packet.data\_field\_size = size;

d\_packet.sequence = sequence;

int i = 0;

// control

d\_packet.raw\_bytes = (unsigned char \*)malloc(i + 1);

d\_packet.raw\_bytes[i++] = d\_packet.control;

d\_packet.raw\_bytes = (unsigned char \*)realloc(d\_packet.raw\_bytes, (i + 1));

// sequence

d\_packet.raw\_bytes[i++] = d\_packet.sequence;

d\_packet.raw\_bytes = (unsigned char \*)realloc(d\_packet.raw\_bytes, (i + 1));

// size

unsigned int x = (unsigned int)size;

unsigned char high = (unsigned char)(x >> 8);

unsigned char low = x & 0xff;

d\_packet.raw\_bytes[i++] = high;

d\_packet.raw\_bytes = (unsigned char \*)realloc(d\_packet.raw\_bytes, (i + 1));

d\_packet.raw\_bytes[i++] = low;

d\_packet.raw\_bytes = (unsigned char \*)realloc(d\_packet.raw\_bytes, (i + 1));

// data

for (int j = 0; j < size; j++) {

d\_packet.data[j] = buffer[j];

d\_packet.raw\_bytes[i++] = buffer[j];

d\_packet.raw\_bytes = (unsigned char \*)realloc(d\_packet.raw\_bytes, (i + 1));

}

d\_packet.raw\_bytes\_size = i;

return d\_packet;

}

int main(int argc, char \*argv[]) {

if (argc != 2) {

printf("Usage: %s <number for serial port>\n", argv[0]);

printf("\nExample: %s 11\t-\tfor '/dev/ttyS11'\n", argv[0]);

return -1;

}

/\* opens transmiter file descriptor on second layer \*/

int transmiter\_fd = llopen(atoi(argv[1]), TRANSMITTER);

/\* in case there's an error oppening the port \*/

if (transmiter\_fd == ERROR) {

exit(ERROR);

}

fp = fopen("pinguim.gif", "rb");

file.name = "pinguim.gif";

file.size = get\_file\_size(fp);

file.data = read\_file(fp, file.size);

control\_packet\_t c\_packet\_start = generate\_control\_packet(START);

control\_packet\_t c\_packet\_stop = generate\_control\_packet(STOP);

// sending control packet

struct timespec start;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

print\_control\_packet(c\_packet\_start);

int size = llwrite(transmiter\_fd, c\_packet\_start.raw\_bytes, c\_packet\_start.raw\_bytes\_size);

if (size == ERROR) {

printf("Error writing START Control Packet, aborting...\n");

llclose(transmiter\_fd, TRANSMITTER);

return ERROR;

}

print\_elapsed\_time(start);

unsigned long bytes\_left = file.size;

int index\_start;

int index\_end = -1;

int sequence = 0;

while (bytes\_left != 0 && index\_end != file.size - 1) {

usleep(STOP\_AND\_WAIT);

index\_start = index\_end + 1;

if (bytes\_left >= 1023) {

index\_end = index\_start + 1023;

} else {

index\_end = index\_start + bytes\_left - 1;

}

bytes\_left -= (index\_end - index\_start) + 1;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

data\_packet\_t data = generate\_data\_packet(split\_file(file.data, index\_start, index\_end), index\_end - index\_start + 1, sequence++);

print\_data\_packet(&data, FALSE);

size = llwrite(transmiter\_fd, data.raw\_bytes, data.raw\_bytes\_size);

if (size == ERROR) {

printf("Error writing Data Packet, aborting...\n");

llclose(transmiter\_fd, TRANSMITTER);

return ERROR;

}

print\_elapsed\_time(start);

}

usleep(STOP\_AND\_WAIT);

print\_control\_packet(c\_packet\_stop);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

size = llwrite(transmiter\_fd, c\_packet\_stop.raw\_bytes, c\_packet\_stop.raw\_bytes\_size);

if (size == ERROR) {

printf("Error writing STOP Control Packet, aborting...\n");

llclose(transmiter\_fd, TRANSMITTER);

return ERROR;

}

usleep(STOP\_AND\_WAIT);

print\_elapsed\_time(start);

/\* resets and closes the receiver fd for the port \*/

llclose(transmiter\_fd, TRANSMITTER);

return OK;

}

## data\_link.c

#include "data\_link.h"

int reetransmit = 1;

extern int flag;

extern int conta;

int send\_supervision\_frame(int fd, unsigned char msg, unsigned char address) {

unsigned char mesh[5];

mesh[0] = FLAG;

mesh[1] = address;

mesh[2] = msg;

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

mesh[4] = FLAG;

int err = write(fd, mesh, 5);

if (!(err == 5))

return ERROR;

return 0;

}

unsigned char receive\_acknowledgement(int fd) {

// ! Remove comments if you want to debug the data being read

int part = 0;

unsigned char rcv\_msg;

unsigned char ctrl;

unsigned char address;

printf("Reading ACK supervision frame...\n");

while (part != 5) {

int rd = read(fd, &rcv\_msg, 1);

if (rd == -1 && errno == EINTR) {

printf("READ failed\n");

return 2;

}

switch (part) {

case 0:

if (rcv\_msg == FLAG) {

part = 1;

// printf("FLAG: 0x%x\n",rcv\_msg);

}

break;

case 1:

if (rcv\_msg == A\_1 || rcv\_msg == A\_3) {

address = rcv\_msg;

part = 2;

// printf("A: 0x%x\n",rcv\_msg);

} else {

if (rcv\_msg == FLAG)

part = 1;

else

part = 0;

}

break;

case 2:

if ((rcv\_msg == C\_RR0) || (rcv\_msg == C\_RR1) || (rcv\_msg == C\_REJ0) ||

(rcv\_msg == C\_REJ1)) {

part = 3;

// printf("Control: 0x%x\n",rcv\_msg);

ctrl = rcv\_msg;

} else

part = 0;

break;

case 3:

if (rcv\_msg == (address ^ ctrl)) {

part = 4;

// printf("Control BCC: 0x%x\n",rcv\_msg);

} else

part = 0;

break;

case 4:

if (rcv\_msg == FLAG) {

part = 5;

// printf("FINAL FLAG: 0x%x\nReceived Control\n",rcv\_msg);

} else

part = 0;

break;

default:

break;

}

}

return ctrl;

}

int receive\_supervision\_frame(int fd, unsigned char msg) {

// ! Remove comments if you want to debug the data being read

int part = 0;

unsigned char rcv\_msg;

unsigned char address;

printf("Reading supervision frame...\n");

while (part != 5) {

int rd = read(fd, &rcv\_msg,1);

if (rd == -1 && errno == EINTR) {

printf("READ failed\n");

return 2;

}

switch (part) {

case 0:

if (rcv\_msg == FLAG) {

part = 1;

// printf("FLAG: 0x%x\n",rcv\_msg);

}

break;

case 1:

if (rcv\_msg == A\_1 || rcv\_msg == A\_3) {

address = rcv\_msg;

part = 2;

// printf("A: 0x%x\n",rcv\_msg);

} else {

if (rcv\_msg == FLAG)

part = 1;

else

part = 0;

}

break;

case 2:

if (rcv\_msg == msg) {

part = 3;

// printf("Control: 0x%x\n",rcv\_msg);

} else

part = 0;

break;

case 3:

if (rcv\_msg == (address ^ msg)) {

part = 4;

// printf("Control BCC: 0x%x\n",rcv\_msg);

} else

part = 0;

break;

case 4:

if (rcv\_msg == FLAG) {

part = 5;

// printf("FINAL FLAG: 0x%x\nReceived Control\n",rcv\_msg);

} else

part = 0;

break;

default:

break;

}

}

return (part == 5) ? 0 : -1;

}

int receive\_set(int fd) {

if (receive\_supervision\_frame(fd, SET) == 0) {

printf("Sending UA reply...\n");

send\_supervision\_frame(fd, UA, A\_3);

}

return 0;

}

int send\_set(int fd) {

struct timespec start;

do {

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

if (send\_supervision\_frame(fd, SET, A\_3) == -1) {

printf("Error writing SET\n");

continue;

}

alarm(TIMEOUT); // activa alarme de 3s

printf("Sent SET frame\n");

flag = 0;

printf("Receiving UA response...\n");

while (!flag) {

if (receive\_supervision\_frame(fd, UA) == 0) {

reetransmit = 0;

break;

}

}

if (flag)

printf("Timed Out - Retrying\n");

print\_elapsed\_time(start);

} while (conta < 4 && flag);

alarm(RESET\_ALARM);

if (conta == 4) {

reetransmit = 2;

printf("Gave up\n");

return -1;

}

return 0;

}

int disconnect\_writer(int fd) {

struct timespec start;

do {

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

if (send\_supervision\_frame(fd, DISC, A\_3) == -1) {

printf("Error writing DISC\n");

continue;

}

alarm(TIMEOUT); // activa alarme de 3s

printf("Sent DISC frame\n");

flag = 0;

printf("Receiving DISC response...\n");

while (!flag) {

if (receive\_supervision\_frame(fd, DISC) == 0) {

reetransmit = 0;

break;

}

}

if (flag)

printf("Timed Out - Retrying\n");

print\_elapsed\_time(start);

} while (conta < 4 && flag);

alarm(RESET\_ALARM);

if (conta == 4) {

reetransmit = 2;

printf("Gave up\n");

return -1;

}

printf("Sending UA ACK...\n");

return send\_supervision\_frame(fd, UA, A\_1);

}

int disconnect\_reader(int fd) {

struct timespec start;

do {

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

alarm(TIMEOUT); // activa alarme de 3s

flag = 0;

printf("Receiving DISC from writer...\n");

while (!flag) {

if (receive\_supervision\_frame(fd, DISC) == 0) {

reetransmit = 0;

break;

}

}

printf("DISC received, sending DISC..\n");

if (send\_supervision\_frame(fd, DISC, A\_3) == -1) {

printf("Error writing DISC\n");

continue;

}

if (flag)

printf("Timed Out - Retrying\n");

print\_elapsed\_time(start);

} while (conta < 4 && flag);

printf("Receiving UA...\n");

return receive\_supervision\_frame(fd, UA);

}

int send\_acknowledgement(int fd, int frame, int accept) {

printf("Sending acknowledgement...\n");

if (frame == 0) {

if (accept == 1) {

// caso seja o frame 0 e seja aceite então pede o frame 1

send\_supervision\_frame(fd, C\_RR1, A\_3);

} else {

send\_supervision\_frame(fd, C\_REJ0, A\_3);

}

} else {

if (accept == 1) {

send\_supervision\_frame(fd, C\_RR0, A\_3);

} else {

send\_supervision\_frame(fd, C\_REJ1, A\_3);

}

}

return 0;

}

int llopen(int port, int type) {

char file[48];

sprintf(file, "/dev/ttyS%d", port);

int fd;

if (type == TRANSMITTER) {

if ((fd = open\_writer(file)) == ERROR) {

perror("llopen: error on open\_writer");

return ERROR;

}

if (send\_set(fd) == ERROR) {

perror("llopen: error sending SET");

return ERROR;

}

return fd;

}

else if (type == RECEIVER) {

if ((fd = open\_reader(file)) == ERROR) {

perror("llopen: error on open\_reader");

return ERROR;

}

if (receive\_set(fd) == ERROR) {

perror("llopen: error receiving SET");

return ERROR;

}

return fd;

}

perror("llopen: type not valid");

return ERROR;

}

int llclose(int fd, int type) {

printf("\nDisconnecting...\n");

if (type == TRANSMITTER) {

if (disconnect\_writer(fd) == ERROR) {

perror("llclose: error disconnecting writer: ");

return ERROR;

}

if (close\_writer(fd) != ERROR) {

printf("Writer Successfully Closed!\n");

return OK;

} else {

perror("llclose: writer not closed successfully: ");

return ERROR;

}

}

else if (type == RECEIVER) {

if (disconnect\_reader(fd) == ERROR) {

perror("llclose: error disconnecting reader: ");

return ERROR;

}

if (close\_reader(fd) != ERROR) {

printf("Reader Successfully Closed!\n");

return OK;

} else {

perror("llclose: reader not closed successfully: ");

return ERROR;

}

}

return ERROR;

}

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

printf("Sending data...\n");

// printf("Message: %s\n", buffer);

printf("Coding message...\n");

information\_frame\_t frame; // to keep everything organized

frame.address = A\_3;

/\* C byte - Controls package, alternating between 0 and 1\*/

frame.control = (current\_frame == 0) ? C\_I0 : C\_I1;

frame.bcc1 = frame.address ^ frame.control;

int size\_info = length;

unsigned char \*information\_frame = (unsigned char \*) malloc (size\_info \*sizeof(unsigned char));

unsigned char bcc = 0xff;

int i = 0;

for (int j = 0; j < length; j++) {

/\* Data stuffing and buffer size adjusting\*/

if (buffer[j] == ESCAPE) {

information\_frame =

(unsigned char \*)realloc(information\_frame, ++size\_info);

information\_frame[i++] = ESCAPE;

information\_frame[i++] = ESCAPE\_ESC;

} else if (buffer[j] == FLAG) {

information\_frame =

(unsigned char \*)realloc(information\_frame, ++size\_info);

information\_frame[i++] = ESCAPE;

information\_frame[i++] = ESCAPE\_FLAG;

} else

information\_frame[i++] = buffer[j];

bcc = buffer[j] ^ bcc;

}

frame.data = information\_frame; /\* saves the stuffed data-buffer on the struct \*/

frame.data\_size = i; /\* size of the suffed data-buffer \*/

frame.bcc2 = bcc; /\* this BCC2 is not stuffed yet and it will be displayed \*unstuffed\* \*/

/\* Saving all data to be transmitted to .raw\_bytes \*/

frame.raw\_bytes = (unsigned char \*)malloc((frame.data\_size + 10) \* sizeof(unsigned char \*));

int j = 0;

frame.raw\_bytes[j++] = FLAG;

frame.raw\_bytes[j++] = frame.address;

frame.raw\_bytes[j++] = frame.control;

frame.raw\_bytes[j++] = frame.bcc1;

for (int k = 0; k < frame.data\_size; k++) {

frame.raw\_bytes[j++] = frame.data[k];

}

/\* BCC2 stuffing\*/

if (bcc == ESCAPE) {

frame.raw\_bytes[j++] = ESCAPE;

frame.raw\_bytes[j++] = ESCAPE\_ESC;

} else if (bcc == FLAG) {

frame.raw\_bytes[j++] = ESCAPE;

frame.raw\_bytes[j++] = ESCAPE\_FLAG;

} else

frame.raw\_bytes[j++] = bcc;

frame.raw\_bytes[j++] = FLAG;

// ! remove next comment if you want to see the coded message being written

// print\_message(frame, TRUE);

conta = 1;

int count = -1;

do {

if ((count = write(fd, frame.raw\_bytes, j)) != ERROR) {

printf("Message sent! Waiting for ACK\n");

} else {

printf("Message not sent!\n");

return ERROR;

// adicionei esta linha, pq caso não escreva corretamente

// deve retornar -1 para escrever de novo

}

alarm(TIMEOUT);

flag = 0;

unsigned char ack = receive\_acknowledgement(fd);

if (ack == C\_REJ0 || ack == C\_REJ1) {

printf("Received negative ACK\n");

alarm(RESET\_ALARM);

return ERROR;

}

// Retransmition

if ((ack == C\_RR0 && current\_frame == 0) ||

(ack == C\_RR1 && current\_frame == 1)) {

printf("Received positive ACK (retransmition)\n");

alarm(RESET\_ALARM);

// returns error but to the application only means it has to

// send the same frame again

return ERROR;

}

if ((ack == C\_RR0 && current\_frame == 1) ||

(ack == C\_RR1 && current\_frame == 0)) {

printf("Received positive ACK\n");

alarm(RESET\_ALARM);

current\_frame = (current\_frame == 0) ? 1 : 0; // changes the current frame

return count;

} else {

printf("Timed out\nTrying again\n");

alarm(RESET\_ALARM);

}

printf("Couldn't receive ACK in time\n");

} while (flag && conta < 4);

return ERROR;

}

int llread(int fd, char \*buffer) {

information\_frame\_t information\_frame;

information\_frame.raw\_bytes = (unsigned char \*)malloc(sizeof(unsigned char));

int i = 0;

int part = 0;

unsigned char rcv\_msg;

printf("Reading...\n");

// \* lógica: processar os dados todos em raw bytes, depois fazer o unstuffing,

// \* e depois fazer o tratamento dos dados

/\*

part 0 - before first flag

part 1 - between flag start and flag stop

part 2 - after flag stop \*/

while (part != 2) {

read(fd, &rcv\_msg, 1);

if (rcv\_msg == FLAG && part == 0) {

part = 1;

continue;

} else if (rcv\_msg == FLAG && part == 1) {

part = 2;

break;

}

information\_frame.raw\_bytes[i++] = rcv\_msg;

information\_frame.raw\_bytes = (unsigned char \*)realloc(information\_frame.raw\_bytes, (i + 1));

}

int data\_size = i;

/\* UNSTUFFING BYTES \*/

int j = 0, p = 0;

for (; j < i && p < i; j++) {

if (information\_frame.raw\_bytes[p] == ESCAPE) {

information\_frame.raw\_bytes =

(unsigned char \*)realloc(information\_frame.raw\_bytes, --data\_size);

if (information\_frame.raw\_bytes[p + 1] == ESCAPE\_ESC)

information\_frame.raw\_bytes[j] = ESCAPE;

else if (information\_frame.raw\_bytes[p + 1] == ESCAPE\_FLAG)

information\_frame.raw\_bytes[j] = FLAG;

p += 2;

} else {

information\_frame.raw\_bytes[j] = information\_frame.raw\_bytes[p];

p++;

}

}

information\_frame.data =

(unsigned char \*)malloc((data\_size - 4) \* sizeof(unsigned char));

information\_frame.address = information\_frame.raw\_bytes[0];

information\_frame.control = information\_frame.raw\_bytes[1];

information\_frame.bcc1 = information\_frame.raw\_bytes[2];

p = 0;

for (int byte = 3; byte < data\_size - 1; byte++) {

information\_frame.data[p++] = information\_frame.raw\_bytes[byte];

}

information\_frame.bcc2 = information\_frame.raw\_bytes[data\_size - 1];

information\_frame.data\_size = data\_size - 4;

// ! remove \*sleep\* comments if you want to check what happens when ACK is not received in time

// ! remove print\_message comment if you want to see the data byte-by-byte

int bccError = verify\_message(information\_frame);

if (bccError == ERROR) {

// sleep(15);

send\_acknowledgement(fd, current\_frame, FALSE);

} else {

// sleep(4);

send\_acknowledgement(fd, current\_frame, TRUE);

current\_frame = (current\_frame == 0) ? 1 : 0;

// print\_message(information\_frame, FALSE);

}

for (i = 0; i < information\_frame.data\_size; i++) {

buffer[i] = information\_frame.data[i];

}

free(information\_frame.raw\_bytes);

free(information\_frame.data);

return (bccError == OK) ? information\_frame.data\_size : ERROR;

}

## data\_link.h

#include "utils.h"

#define TRANSMITTER 0

#define RECEIVER 1

#define STOP\_AND\_WAIT 50

/\*\*

\* current I-Frame

\*/

static int current\_frame = 0;

/\*\*

\* takes file descriptor (port) and sends a code msg in a

\* supervision frame

\*/

int send\_supervision\_frame(int fd, unsigned char msg, unsigned char address);

/\*\*

\* receives a supervision frame with controll as msg

\*/

int receive\_supervision\_frame(int fd, unsigned char msg);

/\*\*

\* receives a ACK frame and returns it's control byte

\*/

unsigned char receive\_acknowledgement(int fd);

int send\_acknowledgement(int fd, int frame, int accept);

/\*\*

\* Reading Fucntion

\* @param fd Serial Port to be read

\*/

int receive\_set(int fd);

/\*\*

\* This function sends a SET Control frame and expects an UA

\*/

int send\_set(int fd);

int disconnect\_writer(int fd);

int disconnect\_reader(int fd);

/\*\*

\* This function opens a port and returns the file descriptor

\* @param port port number to be open

\* @param type RECEIVER or TRANSMITTOR

\*/

int llopen(int port, int type);

/\*\*

\* Same as llopen but this one closes the fd

\*/

int llclose(int fd, int type);

/\*\*

\* @brief Function to write a buffer to a file descriptor

\* This function takes the buffer and sends it trought the port,

\* after the byte-stuffing

\* @returns -1 if error or number of bytes written for success

\*/

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

/\*\*

\* @brief Function to read a buffer from a file descriptor

\* This function reads a buffer from the port and returns the number

\* @returns -1 if error or number of bytes read for success

\*/

int llread(int fd, char \*buffer);

## files.c

#include "files.h"

unsigned long get\_file\_size(FILE\* f) {

fseek(f, 0, SEEK\_END); // seek to end of file

unsigned long size = ftell(f); // get current file pointer

fseek(f, 0, SEEK\_SET); // seek back to beginning of file

// proceed with allocating memory and reading the file

return size;

}

unsigned char\* read\_file(FILE\* f, unsigned long filesize) {

unsigned char\* data = (unsigned char\*) malloc (filesize);

fread(data, sizeof(unsigned char), filesize, f);

return data;

}

unsigned char\* split\_file(unsigned char\* data, unsigned long index\_start, unsigned long index\_end) {

int range = index\_end - index\_start + 1;

unsigned char\* frame = (unsigned char\*) malloc (range);

for (int k = 0; k < range; k++) {

frame[k] = data[index\_start + k];

}

return frame;

}

void join\_file(unsigned char\* data, unsigned char\* frame, unsigned long size, int index) {

for (int j = 0; j < size; j++) {

data[index + j] = frame[j];

}

}

void write\_file(char\* name, unsigned char\* bytes, unsigned long size) {

FILE \*fh = fopen (name, "wb");

if (fh != NULL) {

fwrite (bytes, sizeof (unsigned char), size, fh);

fclose (fh);

}

}int fd, char \*buffer);

## files.h

#include "macros.h"

unsigned long get\_file\_size(FILE\* f);

unsigned char\* read\_file(FILE\* f, unsigned long filesize);

unsigned char\* split\_file(unsigned char\* data, unsigned long index\_start, unsigned long index\_end);

void join\_file(unsigned char\* data, unsigned char\* frame, unsigned long size, int index);

void write\_file(char\* name, unsigned char\* bytes, unsigned long size);

## macros.h

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <termios.h>

#include <time.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include <errno.h>

#define BAUDRATE B38400

#define MODEMDEVICE "/dev/ttyS1"

#define \_POSIX\_SOURCE 1 /\* POSIX compliant source \*/

#define FALSE 0

#define TRUE 1

#define OK 0

#define ERROR -1

#define TRIES 3

#define TIMEOUT 3

#define RESET\_ALARM 0

// Mesh Macros

#define FLAG 0x7E

#define A\_3 0x03

#define A\_1 0x01

#define SET 0x03

#define UA 0x07

#define DISC 0x0B

#define SET\_BCC A ^ SET

#define UA\_BCC A ^ UA

// Control Macros (there is another way of defining them)

#define C\_I0 0x00

#define C\_I1 0x40

#define C\_RR0 0x05

#define C\_RR1 0x85

#define C\_REJ0 0x01

#define C\_REJ1 0x81

//Byte Stuffing

#define ESCAPE 0x7D

#define ESCAPE\_ESC 0x5D

#define ESCAPE\_FLAG 0x5E

// packet Macros

#define DATA 0x1

#define START 0x2

#define STOP 0x3

#define FILE\_SIZE 0x0

#define FILE\_NAME 0x1

#define PACKET\_SIZE 1024

## noncanonical.c

/\*Non-Canonical Input Processing\*/

#include "noncanonical.h"

extern int reetransmit;

static struct termios oldtio;

static struct termios newtio;

int open\_reader(char \*port) {

int fd;

/\* top layer does the verification of the port name \*/

/\*

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.

\*/

fd = open(port, O\_RDWR | O\_NOCTTY);

if (fd < 0) {

perror(port);

return -1;

}

if (tcgetattr(fd, &oldtio) == -1) { /\* save current port settings \*/

perror("tcgetattr");

return -1;

}

bzero(&newtio, sizeof(newtio));

newtio.c\_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;

newtio.c\_iflag = IGNPAR;

newtio.c\_oflag = 0;

/\* set input mode (non-canonical, no echo,...) \*/

newtio.c\_lflag = 0;

newtio.c\_cc[VTIME] = 1; /\* inter-character timer unused \*/

newtio.c\_cc[VMIN] = 0; /\* blocking read until 1 chars received \*/

/\*

VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a

leitura do(s) proximo(s) caracter(es)

\*/

tcflush(fd, TCIOFLUSH);

if (tcsetattr(fd, TCSANOW, &newtio) == -1) {

perror("tcsetattr");

return -1;

}

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

return fd;

}

int close\_reader(int fd) {

if (tcsetattr(fd, TCSANOW, &oldtio) == -1) {

perror("tsetattr");

return -1;

}

close(fd);

return 0;

}

## noncanonical.h

#include "macros.h"

/\*\*

\* @brief Opens the reader

\*/

int open\_reader(char \*port);

/\*\*

\* @brief Reset and close the reader

\*/

int close\_reader(int fd);

## utils.c

#include "utils.h"

control\_packet\_t parse\_control\_packet(unsigned char \*raw\_bytes, int size) {

control\_packet\_t packet;

memset(&packet, 0, sizeof(control\_packet\_t));

packet.control = raw\_bytes[0];

char \*name;

int namesize = 0;

unsigned char\* filesize;

int filesize\_size = 0;

for (int i = 1; i < size; i++) {

if (raw\_bytes[i] == FILE\_SIZE) {

int length = raw\_bytes[++i];

int offset = i + length;

filesize = (unsigned char\*) malloc (length);

for (int k = 0; i < offset; k++) {

filesize[k] = raw\_bytes[++i];

filesize\_size++;

}

continue;

}

if (raw\_bytes[i] == FILE\_NAME) {

int length = raw\_bytes[++i];

name = (unsigned char \*) malloc (length);

int offset = i + length;

for (int j = 0; i < offset;) {

name[j++] = raw\_bytes[++i];

namesize++;

}

continue;

}

}

packet.file\_name = (unsigned char\*) malloc (namesize + 1);

memcpy(packet.file\_name, name, namesize);

packet.file\_name[namesize] = '\0';

free(name);

packet.filesize\_size = filesize\_size;

packet.file\_size = (unsigned char\*) malloc (filesize\_size);

memcpy(packet.file\_size, filesize, filesize\_size);

free(filesize);

return packet;

}

data\_packet\_t parse\_data\_packet(unsigned char \*raw\_bytes, int size) {

data\_packet\_t packet;

memset(&packet, 0, sizeof(data\_packet\_t));

packet.raw\_bytes\_size = size;

packet.control = raw\_bytes[0];

packet.sequence = raw\_bytes[1];

packet.data\_field\_size = (raw\_bytes[2] << 8) | raw\_bytes[3];

for (int i = 0; i < packet.data\_field\_size; i++) {

packet.data[i] = raw\_bytes[4 + i];

}

return packet;

}

void print\_control\_packet(control\_packet\_t packet) {

printf("---- CONTROL PACKET ----\n");

switch (packet.control) {

case START:

printf("Control: START (0x%x)\n", packet.control);

break;

case STOP:

printf("Control: STOP (0x%x)\n", packet.control);

break;

default:

break;

}

printf("Size: %ld %#lx\n", array\_to\_number(packet.file\_size, packet.filesize\_size), array\_to\_number(packet.file\_size, packet.filesize\_size));

printf("Name: %s\n", packet.file\_name);

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

}

void print\_data\_packet(data\_packet\_t\* packet, int full\_info) {

printf("---- DATA PACKET ----\n");

printf("Control: - (0x%x)\n", packet->control);

printf("Data size: %d (0x%x)\n", packet->data\_field\_size,

packet->data\_field\_size);

printf("Sequence: %d (0x%x)\n", packet->sequence, packet->sequence);

if (full\_info) {

for (int i = 0; i < packet->data\_field\_size; i++) {

printf("DATA[%d]: %c (0x%x)\n", i, packet->data[i], packet->data[i]);

}

}

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

}

void print\_message(information\_frame\_t frame, int stuffed) {

printf("Address: 0x%x\n", frame.address);

printf("Control: 0x%x\n", frame.control);

printf("BCC1: 0x%x\n", frame.bcc1);

int j = 0;

for (int i = 0; i < frame.data\_size; i++) {

if (frame.data[i] == ESCAPE && stuffed) {

printf("DATA[%d]: 0x%x - ESCAPE\n", j++, frame.data[i++]);

if (frame.data[i] == ESCAPE\_ESC) {

printf("DATA[%d]: 0x%x - ESCAPED ESCAPE\n", j++, frame.data[i]);

} else if (frame.data[i] == ESCAPE\_FLAG) {

printf("DATA[%d]: 0x%x - ESCAPED FLAG\n", j++, frame.data[i]);

}

} else {

printf("DATA[%d]: 0x%x - %c (char)\n", j++, frame.data[i], frame.data[i]);

}

}

printf("BCC2: 0x%x\n", frame.bcc2);

printf("Message: %s - size: %d - strlen: %ld\n", frame.data, frame.data\_size,

strlen(frame.data));

}

int verify\_message(information\_frame\_t frame) {

if (frame.bcc1 != (frame.control ^ frame.address)) {

return ERROR;

}

unsigned char bcc2 = 0xff;

for (int i = 0; i < frame.data\_size; i++) {

bcc2 = frame.data[i] ^ bcc2;

}

if (bcc2 != frame.bcc2) {

return ERROR;

}

return OK;

}

void print\_elapsed\_time(struct timespec start) {

struct timespec end;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

double delta = (end.tv\_sec - start.tv\_sec) \* 1000.0 +

(end.tv\_nsec - start.tv\_nsec) / 1000000.0;

printf("Elapsed time: %f s\n\n", delta);

}

int check\_connection(int fd) {

if (fcntl(fd, F\_GETFD) == -1) {

printf("Connection closed\n");

return -1;

}

return 0;

}

unsigned long array\_to\_number(unsigned char\* buffer, unsigned int size) {

unsigned long value = 0;

int offset = 0;

for (int i = 0; i < size; i++) {

value |= buffer[i] << (8 \* offset++);

}

return value;

}

unsigned int number\_to\_array(unsigned long num, unsigned char\* buffer) {

unsigned int size = 0;

for (int i = 0; i < sizeof(unsigned long); i++) {

buffer[i] = (num >> (8 \* i)) & 0xff;

size += 1;

}

for (int i = sizeof(unsigned long) - 1; i != 0; i--) {

if (buffer[i] != 0) break;

size--;

}

return size;

}

## utils.h

#include "macros.h"

#include "writenoncanonical.h"

#include "noncanonical.h"

#include "app\_structs.h"

/\*\*

\* @brief This function parses a buffer of raw bytes to a control\_packet\_t

\* structure

\* This method takes the raw\_bytes and parses them into a control packet

\*/

control\_packet\_t parse\_control\_packet(unsigned char \*raw\_bytes, int size);

/\*\*

\* @brief This function parses a buffer of raw bytes to a data\_packet\_t

\* structure

\* This method takes the raw\_bytes and parses them into a data packet

\*/

data\_packet\_t parse\_data\_packet(unsigned char \*raw\_bytes, int size);

/\*\*

\* @brief This function pretty-prints a control packet

\*/

void print\_control\_packet(control\_packet\_t packet);

/\*\*

\* @brief This function pretty-prints a data packet

\*/

void print\_data\_packet(data\_packet\_t\* packet, int full\_info);

/\*\*

\* @brief Method to pretty-print the elapsed time between two frames

\*/

void print\_elapsed\_time(struct timespec start);

/\*\*

\* @brief Method to verify an I-Frame

\* Checks if there are errors on the BCC bytes

\* @returns 0 if no error or -1 for error

\*/

int verify\_message(information\_frame\_t frame);

/\*\*

\* @brief Method to pretty-print an information frame details

\*/

void print\_message(information\_frame\_t frame, int coded);

/\*\*

\* @brief Takes fd and checks if it hasn't closed

\*/

int check\_connection(int fd);

/\*\*

\* @brief takes an array with length size, and converts it to an 8byte number

\* array[0] = MSB ; array[size - 1] = LSB

\*/

unsigned long array\_to\_number(unsigned char\* buffer, unsigned int size);

/\*\*

\* @brief Method to convert a 8byte number into an 8 byte char array

\* array[0] = MSB ; array[return - 1] = LSB

\* @return array's size

\*/

unsigned int number\_to\_array(unsigned long num, unsigned char\* buffer);

## writenoncanonical.c

/\*Non-Canonical Input Processing\*/

#include "writenoncanonical.h"

int flag = 1, conta = 1;

extern int reetransmit;

static struct termios oldtio;

static struct termios newtio;

void atende() { // atende alarme

printf("alarme # %d\n", conta);

flag = 1;

conta++;

}

int open\_writer(char \*port) {

/\* top level layer should verifiy ports name \*/

/\*

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.

\*/

int fd = open(port, O\_RDWR | O\_NOCTTY);

if (fd < 0) {

perror(port);

return -1;

}

if (tcgetattr(fd, &oldtio) == -1) {

/\* save current port settings \*/

perror("tcgetattr");

return -1;

}

bzero(&newtio, sizeof(newtio));

newtio.c\_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;

newtio.c\_iflag = IGNPAR;

newtio.c\_oflag = 0;

/\* set input mode (non-canonical, no echo,...) \*/

newtio.c\_lflag = 0;

newtio.c\_cc[VTIME] = TIMEOUT; /\* inter-character timer unused \*/

newtio.c\_cc[VMIN] = 0; /\* blocking read 1 char received \*/

/\*

VTIME e VMIN devem ser alterados de forma a proteger com um

temporizador a

leitura do(s) pr�ximo(s) caracter(es)

\*/

tcflush(fd, TCIOFLUSH);

if (tcsetattr(fd, TCSANOW, &newtio) == -1) {

perror("tcsetattr");

return -1;

}

/\*struct sigaction action;

sigemptyset(&action.sa\_mask);

action.sa\_handler = atende;

action.sa\_flags = 0;

sigaction(SIGALRM,&action,NULL);\*/ // instala rotina que atende interrupcao

signal(SIGALRM, atende);

siginterrupt(SIGALRM, 1);

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

return fd;

}

int close\_writer(int fd) {

if (tcsetattr(fd, TCSANOW, &oldtio) == -1) {

perror("tcsetattr");

return -1;

}

close(fd);

return 0;

}

## writenoncanonical.h

#include "macros.h"

/\*\*

\* function to open and set port

\*/

int open\_writer(char \*port);

/\*\*

\* function to reset and close port

\*/

int close\_writer(int fd);