

**Redes de Computadores**

PROTOCOLO DE LIGAÇÃO DE DADOS

3MIEIC04 - Grupo 7

**Diogo Almeida –** [**up201806630@fe.up.pt**](mailto:up201806630@fe.up.pt)

**Pedro Queirós –** [**up201806329@fe.up.pt**](mailto:up201806329@fe.up.pt)

Índice

[Sumário 2](#_Toc56116016)

[Introdução 2](#_Toc56116017)

[Arquitetura 3](#_Toc56116018)

[Estrutura do Código 3](#_Toc56116019)

[ll.c / ll.h 3](#_Toc56116020)

[application.c / application.h 4](#_Toc56116021)

[utils.h 4](#_Toc56116022)

[proj.c 5](#_Toc56116023)

[Casos de Uso Principais 5](#_Toc56116024)

[Protocolo de Ligação Lógica 5](#_Toc56116025)

[llopen 5](#_Toc56116026)

[llwrite 6](#_Toc56116027)

[llread 6](#_Toc56116028)

[llclose 6](#_Toc56116029)

[Protocolo de aplicação 6](#_Toc56116030)

[sendFile 6](#_Toc56116031)

[receiveFile 7](#_Toc56116032)

[Validação 7](#_Toc56116033)

[Eficiência do Protocolo de Ligação de Dados 8](#_Toc56116034)

[Conclusões 9](#_Toc56116035)

[Anexos 10](#_Toc56116036)

[Anexo I – Código fonte 10](#_Toc56116037)

[application.h 10](#_Toc56116038)

[application.c 11](#_Toc56116039)

[ll.h 16](#_Toc56116040)

[ll.c 17](#_Toc56116041)

[utils.h 28](#_Toc56116042)

[proj.c 29](#_Toc56116043)

[Anexo II – Resultados em Ambiente de Laboratório 31](#_Toc56116044)

# Sumário

Este trabalho foi desenvolvido no âmbito da unidade curricular de Redes de Computadores com o objetivo de implementar e analisar um protocolo de ligação de dados. Este protocolo consiste na transferência de dados entre computadores através da porta série.

O trabalho foi concluído com sucesso, visto que a aplicação desenvolvida estabelece a ligação entre dois computadores eficientemente sem qualquer perda de dados.

# Introdução

O objetivo do trabalho consiste na criação de software, segundo o guião fornecido, que permita estabelecer um protocolo de ligação de dados, ou seja, a transferência de ficheiros entre dois computadores.

Este relatório, com o objetivo de detalhar a componente teórica do projeto, aborda os seguintes temas:

* **Arquitetura**
  + Blocos funcionais e interfaces
* **Estrutura do código**
  + APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura
* **Casos de usos principais**
  + Identificação
  + Sequências de chamada de funções
* **Protocolo de ligação lógica**
  + Identificação dos principais aspetos funcionais
  + Descrição da estratégia de implementação destes aspetos
* **Protocolo de aplicação**
  + Identificação dos principais aspetos funcionais
  + Descrição da estratégia de implementação destes aspetos
* **Validação**
  + Descrição dos testes efetuados com apresentação quantificada dos resultados, se possível
* **Eficiência do protocolo de ligação de dados**
  + Caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido
* **Conclusões**
  + Síntese da informação apresentada nas secções anteriores
  + Reflexão sobre os objetivos de aprendizagem alcançados

# Arquitetura

O software que implementa o protocolo de ligação de dados está divido em dois módulos independentes entre si: o módulo que estabelece a **ligação de dados** e o módulo da **aplicação**.

O módulo da ligação de dados trata de toda a comunicação com a porta série, isto é, a sua abertura e fecho, bem como a sua leitura e escrita. Além disso, este módulo é também responsável pela criação e processamento de cada trama, tratando da sua delimitação, transparência, proteção e retransmissão em caso de erro.

O módulo da aplicação, utilizando o módulo de ligação de dados, é responsável pelo envio e receção de pacotes, quer de controlo quer de informação. Cada pacote é ainda estruturado por este mesmo módulo, efetuando o tratamento do seu cabeçalho e definindo a sua numeração.

A **independência** entre módulos/camadas é assegurada pelos seguintes factos:

* No módulo de ligação de dados não é efetuado qualquer tipo de distinção entre pacotes de controlo e de dados, nem é tida em conta a numeração dos pacotes;
* No módulo da aplicação não há conhecimento acerca do método de ligação de dados, ou seja, este módulo desconhece a estruturação das tramas, a sua criação, e o *stuffing*/*destuffing* a que estão sujeitas. No entanto, tem acesso às funções do módulo de ligação de dados, para o envio e receção de informação.

# Estrutura do Código

Para o desenvolvimento do protocolo de ligação de dados foram utilizadas diversas funções e estruturas de dados em cada um dos módulos.

## ll.c / ll.h

* **checkBaudrate –** Função que converte a *baudrate* de *string* para o respetivo valor (*speed\_t*);
* **ConnectionInfo –** *Struct* que contém informação relativa à conexão entre os dois computadores: número de tentativas, *flag* do alarme, número de sequência da trama, baudrate e *flag* que sinaliza o envio da primeira trama;
* **infoSetup –** Função que inicializa corretamente os valores dos parâmetros da *Struct ConnectionInfo;*
* **verifyControlByte –** Função que verifica se o byte de controlo está correto;
* **responseStateMachine –** Função que implementa a máquina de estados que processa as respostas por parte do emissor e do transmissor;
* **informationFrameStateMachine –** Função que implementa a máquina de estados que processa as tramas de informação enviadas pelo transmissor e recebidas pelo recetor;
* **verifyFrame –** Função que verifica se existem erros na trama recebida através dos bytes de controlo de erros, designados *bcc1* e *bcc2*;
* **processControlByte –** Função que processa a resposta do recetor, indicando se este recebeu a trama corretamente ou não;
* **readReceiverResponse –** Função que lê a resposta do recetor;
* **readTransmitterResponse –** Função que lê a resposta do emissor;
* **readTransmitterFrame –** Função que lê a trama de informação enviada pelo emissor;
* **llopen –** Função que abre a ligação da porta série do emissor/recetor e inicializa a ligação de dados;
* **llwrite –** Função que efetua o *stuffing*, criação da trama e envio desta para o recetor. Após a transmissão, aguarda uma resposta do emissor, agindo em conformidade com a mesma;
* **llread –** Função que efetua a leitura da trama, *destuffing* e envia a resposta para o transmissor de acordo com a trama recebida;
* **llclose –** Função que termina a ligação de dados encerrando, de seguida, a porta série do emissor/recetor;
* **sigAlarmHandler –** Função que processa o sinal após o alarme ter sido ativado;
* **initializeAlarm –** Função que inicializa o alarme;
* **disableAlarm –** Função que desativa o alarme.

## application.c / application.h

* **ApplicationLayer –** *Struct*, utilizada pelo emissor*,* que armazena as informações do módulo da aplicação: descritor do ficheiro, descritor da porta série, tamanho do ficheiro em bytes, nome do ficheiro e tamanho do pacote de dados;
* **ControlPacketInformation –** *Struct, utilizada pelo recetor,* que armazena o nome do ficheiro e o tamanho do mesmo;
* **applicationSetUp –** Função que inicializa corretamente os parâmetros da *Struct* ApplicationLayer;
* **readFileInformation –** Função que abre o ficheiro a ser transmitido guardando as informações relativas ao mesmo;
* **sendControlPacket –** Função que envia cria o pacote de controlo a ser enviado;
* **sendDataPacket –** Função que lê do ficheiro a ser transmitido o número de bytes fornecido pelo utilizador, montando o pacote de dados e enviando para a porta série através da ligação de dados;
* **sendFile –** Função que envia o ficheiro a ser transmitido;
* **readStartControlPacket –** Função que lê o pacote de controlo de inicialização através da ligação de dados;
* **processDataPackets –** Função que escreve o conteúdo do ficheiro no recetor**;**
* **readEndControlPacket –** Função que lê o pacote de controlo de finalização e verifica se o seu conteúdo relativo ao ficheiro é igual ao do pacote de controlo de inicialização recebido anteriormente;
* **receiveFile -** Função que recebe a informação do ficheiro que está a ser transmitido.

## utils.h

Este ficheiro contém todas as macros utilizadas ao longo do programa.

## proj.c

Este ficheiro contém a função main, partilhada pelo recetor e transmissor.

# Casos de Uso Principais

Neste trabalho estão presentes dois casos de uso principais: a interface, no qual o utilizador pode escolher o ficheiro a ser transmitido, e a porta série que estabelece a comunicação entre os dois computadores envolvidos.

Após a compilação do programa, o utilizador deverá passar como argumentos do programa qual a porta série a utilizar, identificação relativa ao emissor/recetor, o ficheiro a ser transmitido, o tamanho do pacote de dados e a *baudrate* a que vai ser transmitido.

Sequência de chamadas de funções por parte do transmissor:

* abertura da ligação de dados através da função *llopen*;
* leitura do nome e do tamanho do ficheiro a ser transmitido através da função *readFileInformation*;
* criação e envio do pacote de controlo START através da função *sendControlPacket*;
* criação e envio dos pacotes de dados através da função *sendDataPacket*;
* criação e envio do pacote de controlo END através da função *sendControlPacket*;
* fecho da ligação de dados através da função *llclose.*

Sequência de chamadas de funções por parte do recetor:

* abertura da ligação de dados através da função *llopen*;
* leitura, efetuada pela função *receiveFile*, e processamento do pacote de controlo START através da função *readStartControlPacket*;
* leitura, efetuada pela função *receiveFile*, processamento dos pacotes de dados através da função *processDataPackets*;
* leitura, efetuada pela função *receiveFile*, processamento do pacote de controlo END através da função *readEndControlPacket*;
* fecho da ligação de dados através da função *llclose*;

# Protocolo de Ligação Lógica

## llopen

Esta função é responsável pelo estabelecimento da ligação entre os dois computadores envolvidos. Assim sendo, quer no transmissor quer no recetor, efetua a abertura da porta série pela qual será transmitida a informação.

No caso do transmissor é ainda enviado numa trama de controlo um SET, indicando ao recetor que será iniciada a transferência de dados, esperando pela confirmação deste através de uma trama de controlo com um UA. Se essa confirmação não chegar ao fim de 20 segundos, ocorre um TIMEOUT e é reenviada a trama de controlo contendo o SET e o processo inicia se de novo.

No caso do recetor é recebido o SET numa trama de controlo e enviada a confirmação de que a transferência de dados poderá começar através de uma trama de controlo contendo um UA.

## llwrite

Esta função é responsável pelo envio de tramas de informação.

Primeiramente esta função cria o cabeçalho da trama a ser enviada. De seguida, é feito o *stuffing* da mensagem a ser enviada assim como o cálculo do *bcc2*. Após este processo, é ainda efetuado o *stuffing* do *bcc2*. Por fim, a trama de informação é enviada para a porta série, ficando à espera da resposta do recetor. Caso esta resposta seja negativa, NACK, ou ocorra um TIMEOUT, a função reenvia a trama de informação. Caso contrário, a função termina corretamente uma vez que recebeu uma resposta positiva, ACK.

## llread

Esta função é responsável pela receção de tramas de informação.

Primeiramente, é efetuada a leitura byte a byte da porta série. De seguida, é efetuado o *destuffing* do campo de dados da trama de informação recebida. Depois, são analisados os campos de proteção, ou seja, o *bcc1* e *bcc2* para ser determinada a resposta a ser enviada para o transmissor. Em caso de algum erro ser detetado, é enviado um NACK, uma resposta negativa. Em caso de não terem sido detetados erros, é enviado um ACK, uma resposta positiva e o conteúdo do campo de dados é guardado num *array* passado como parâmetro da função.

## llclose

Esta função é responsável pela terminação da ligação entre os dois computadores envolvidos.

No caso do transmissor, é enviada uma trama de controlo contendo um DISC, indicando que a ligação de dados será terminada, ficando à espera de receber uma trama de controlo contendo um DISC por parte do recetor. Se ao fim de 20 segundos essa trama não chegar, irá ocorrer um TIMEOUT e o processo inicia-se de novo. Após a receção do DISC, é enviada uma trama de controlo contendo um UA, terminando de seguida a ligação de dados.

No caso do recetor, é recebida uma trama de controlo contendo um DISC, enviada uma trama semelhante à recebida e, por fim, recebida uma trama de controlo contendo UA. Após a receção da última trama de controlo, a ligação de dados é terminada.

# Protocolo de aplicação

Para a implementação do protocolo da aplicação foram implementadas duas funções principais: *sendFile*, no caso do transmissor, e *receiveFile*, no caso do recetor.

## sendFile

Esta função executa todos os procedimentos necessários à transmissão do ficheiro.

Primeiramente, com a ajuda da função *readFileInformation*, armazena a informação relativa ao ficheiro a ser enviado, ou seja, o seu nome e tamanho. De seguida, envia um pacote de controlo START através da função *sendControlPacket*. Este pacote de controlo contém o nome e o tamanho do ficheiro. Após o envio deste pacote, são enviados os pacotes de dados contendo o conteúdo do ficheiro a ser transmitido, utilizando para isso a função *sendDataPackets*. Após o envio do último pacote de dados, é enviado o pacote de controlo END, novamente através da função *sendControlPacket*, contendo também o tamanho e nome do ficheiro transmitido. Todas as funções que enviam um pacote utilizam a função *llwrite* da ligação de dados para o escrever na porta série.

## receiveFile

Esta função efetua a leitura dos pacotes de dados, através da função *llread* da ligação de dados, e o seu processamento.

Ao receber um pacote de controlo START é criado um ficheiro com o nome contido nesse pacote de controlo, utilizando para isso a função *readStartControlPacket*. De seguida, são recebidos os pacotes de dados, processados e a informação relativa ao ficheiro é escrita no ficheiro criado anteriormente através da função *processDataPackets*. Ao receber um pacote de controlo END, a função *readEndControlPacket* irá verificar se o conteúdo deste pacote é igual ao do pacote de controlo *Start* recebido inicialmente, isto é, verificar se o tamanho e nome do ficheiro é igual nos dois pacotes, terminando assim a receção do ficheiro transmitido.

# Validação

Para testar o funcionamento correto do programa, este foi sujeito aos seguintes testes:

* Envio de ficheiros de diversos tamanhos;
* Envio de um mesmo ficheiro com pacotes de tamanhos diferentes;
* Envio de um mesmo ficheiro com várias *baudrates*;
* Interrupção da ligação da porta série durante o envio do ficheiro;
* Introdução de ruído na porta série durante o envio do ficheiro.

O programa concluiu com sucesso todos os testes expostos anteriormente.

# Eficiência do Protocolo de Ligação de Dados

Foram efetuados testes em ambiente virtual e em ambiente físico. Contudo, neste relatório apenas são apresentados dados dos testes efetuados nos laboratórios da FEUP, ou seja, em ambiente físico.

Os testes efetuados à eficiência do programa revelaram que esta é afetada significativamente pelo tamanho dos pacotes de dados. Com os dados apresentados no anexo II, conclui-se que a eficiência do protocolo aumenta à medida que o tamanho dos pacotes de dados aumenta. Contudo, este aumento não é representado por uma função linear, mas sim por uma função semelhante à função logarítmica - até pacotes de 256 bytes a eficiência aumenta significativamente (de 37% para 70%), tendo incrementos mais baixos (apenas de 70% para 75%) a partir deste valor.

O aumento do FER (*Frame Error Ratio*) influência também de forma significativa a eficiência do protocolo de ligação de dados. À medida que a frequência de erros no bcc1 e no bcc2 aumenta, a eficiência do programa diminui.

# Conclusões

Este projeto teve como propósito a implementação de um protocolo de ligação de dados que permitisse a transferência de informação entre dois sistemas através de um meio de comunicação, nomeadamente, a porta série. Após a sua implementação, este programa foi também sujeito a vários testes ao seu desempenho e eficiência que verificaram o seu correto funcionamento.

Em suma, a criação do protocolo de ligação de dados foi bem sucedida, cumprindo todos os objetivos propostos no guião. Ao longo do desenvolvimento do software, foram também adquiridos importantes conhecimentos teórico-práticos em relação ao tema abordado, aprofundando a aprendizagem de conceitos como a independência entre camadas/módulos do programa e o protocolo *Stop&Wait*.

# Anexos

## Anexo I – Código fonte

### application.h

#include <sys/types.h>

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include "ll.h"

**typedef** struct**{**

int fdFile**;**

int fdPort**;**

int fileSize**;**

char**\*** fileName**;**

int packetSize**;**

**}** ApplicationLayer**;**

**typedef** struct

**{**

char**\*** fileName**;**

int fileSize**;**

**}** ControlPacketInformation**;**

void applicationSetUp**(**char **\*** fileName**,** int packetSize**,** int fdPort**);**

int readFileInformation**(**char**\*** fileName**);**

int sendControlPacket**(**unsigned char controlByte**);**

int sendDataPacket**();**

int sendFile**();**

int readStartControlPacket**(**unsigned char **\*** packet**);**

int processDataPackets**(**unsigned char**\*** packet**);**

void readEndControlPacket**(**unsigned char**\*** packet**);**

int receiveFile**();**

### application.c

#include "application.h"

ApplicationLayer app**;**

ControlPacketInformation packetInfo**;**

void applicationSetUp**(**char **\*** fileName**,** int packetSize**,** int fdPort**){**

app**.**fileName **=** fileName**;**

app**.**packetSize **=** packetSize**;**

app**.**fdPort **=** fdPort**;**

**}**

int readFileInformation**(**char**\*** fileName**){**

int fd**;**

struct stat status**;**

**if((**fd **=** open**(**fileName**,**O\_RDONLY**))** **<** 0**){**

perror**(**"Error opening file!\n"**);**

**return** **-**1**;**

**}**

**if(**stat**(**fileName**,&**status**)** **<** 0**){**

perror**(**"Error reading file information!\n"**);**

**return** **-**1**;**

**}**

app**.**fdFile **=** fd**;**

app**.**fileName **=** fileName**;**

app**.**fileSize **=** status**.**st\_size**;**

**return** 0**;**

**}**

int sendControlPacket**(**unsigned char controlByte**){**

int packetIndex **=** 0**;**

int numBytesFileSize **=** **sizeof(**app**.**fileSize**);**

unsigned char packet**[**5**+**numBytesFileSize **+** strlen**(**app**.**fileName**)];**

packet**[**packetIndex**]** **=** controlByte**;**

packetIndex**++;**

packet**[**packetIndex**]** **=** FILE\_NAME\_FLAG**;** //flag nome do ficheiro

packetIndex**++;**

packet**[**packetIndex**]** **=** strlen**(**app**.**fileName**);** // lenght do nome do ficheiro

packetIndex**++;**

**for** **(**size\_t i **=** 0**;** i **<** strlen**(**app**.**fileName**);** i**++)**

**{**

packet**[**packetIndex**]** **=** app**.**fileName**[**i**];**

packetIndex**++;**

**}**

packet**[**packetIndex**]** **=** FILE\_SIZE\_FLAG**;** //flag que indica o tamanho do ficheiro

packetIndex**++;**

packet**[**packetIndex**]** **=** numBytesFileSize**;**

packetIndex**++;**

packet**[**packetIndex**]** **=** **(**app**.**fileSize **>>** 24**)** **&** BYTE\_MASK**;**

packetIndex**++;**

packet**[**packetIndex**]** **=** **(**app**.**fileSize **>>** 16**)** **&** BYTE\_MASK**;**

packetIndex**++;**

packet**[**packetIndex**]** **=** **(**app**.**fileSize **>>** 8**)** **&** BYTE\_MASK**;**

packetIndex**++;**

packet**[**packetIndex**]** **=** app**.**fileSize **&** BYTE\_MASK**;**

packetIndex**++;**

**if(**llwrite**(**app**.**fdPort**,**packet**,**packetIndex**)** **<** packetIndex**){**

printf**(**"Error writing control packet to serial port!\n"**);**

**return** **-**1**;**

**}**

**return** 0**;**

**}**

int sendDataPacket**(){**

int numPacketsSent **=** 0**;**

int numPacketsToSend **=** app**.**fileSize**/**app**.**packetSize**;** // numero máximo de de octetos num packet

unsigned char buffer**[**app**.**packetSize**];**

int bytesRead **=** 0**;**

int length **=** 0**;**

**if(**app**.**fileSize**%**app**.**packetSize **!=** 0**){**

numPacketsToSend**++;**

**}**

**while(**numPacketsSent **<** numPacketsToSend**){**

**if((**bytesRead **=** read**(**app**.**fdFile**,**buffer**,**app**.**packetSize**))** **<** 0**){**

printf**(**"Error reading file\n"**);**

**}**

unsigned char packet**[**4**+**app**.**packetSize**];**

packet**[**0**]** **=** DATA\_FLAG**;**

packet**[**1**]** **=** numPacketsSent **%** 255**;**

packet**[**2**]** **=** bytesRead **/** 256**;**

packet**[**3**]** **=** bytesRead **%** 256**;**

memcpy**(&**packet**[**4**],**buffer**,**bytesRead**);**

length **=** bytesRead **+** 4**;**

**if(**llwrite**(**app**.**fdPort**,**packet**,**length**)** **<** length**){**

printf**(**"Error writing data packet to serial port!\n"**);**

**return** **-**1**;**

**}**

numPacketsSent**++;**

**}**

**return** 0**;**

**}**

int sendFile**(){**

**if(**readFileInformation**(**app**.**fileName**)** **<** 0**){**

printf**(**"Error reading file information!\n"**);**

**return** **-**1**;**

**}**

**if(**sendControlPacket**(**START\_FLAG**)** **<** 0**){**

printf**(**"Error sending start control packet!\n"**);**

**return** **-**1**;**

**}**

**if(**sendDataPacket**()** **<** 0**){**

printf**(**"Error sending data packet!\n"**);**

**return** **-**1**;**

**}**

**if(**sendControlPacket**(**END\_FLAG**)** **<** 0**){**

printf**(**"Error sending end control packet!\n"**);**

**return** **-**1**;**

**}**

**return** 0**;**

**}**

int readStartControlPacket**(**unsigned char **\*** packet**){**

int packetIndex **=** 1**;**

int fileNameLength **=** 0**;**

int fileSize **=** 0**;**

char**\*** fileName**;**

**if(**packet**[**packetIndex**]** **==** FILE\_NAME\_FLAG**){** //flag do nome do ficheiro

packetIndex**++;**

fileName **=** **(**char**\*)** malloc**(**packet**[**packetIndex**]+**1**);**

fileNameLength **=** packet**[**packetIndex**];**

packetIndex**++;**

**for** **(**size\_t i **=** 0**;** i **<** fileNameLength**;** i**++)**

**{**

fileName**[**i**]** **=** packet**[**packetIndex**];**

packetIndex**++;**

**if(**i **==** fileNameLength**-**1**){**

fileName**[**fileNameLength**]** **=** '\0'**;**

**}**

**}**

**}**

**if(**packet**[**packetIndex**]** **==** FILE\_SIZE\_FLAG**){**

packetIndex**+=**2**;**

fileSize **+=** packet**[**packetIndex**]** **<<** 24**;**

packetIndex**++;**

fileSize **+=** packet**[**packetIndex**]** **<<** 16**;**

packetIndex**++;**

fileSize **+=** packet**[**packetIndex**]** **<<** 8**;**

packetIndex**++;**

fileSize **+=** packet**[**packetIndex**];**

**}**

packetInfo**.**fileName **=** fileName**;**

packetInfo**.**fileSize **=** fileSize**;**

app**.**fdFile **=** open**(**fileName**,**O\_WRONLY **|** O\_CREAT **|** O\_APPEND**,** 0664**);**

**return** 0**;**

**}**

void readEndControlPacket**(**unsigned char**\*** packet**){**

int packetIndex **=** 1**;**

int fileNameLength **=** 0**;**

int fileSize **=** 0**;**

char**\*** fileName**;**

**if(**packet**[**packetIndex**]** **==** FILE\_NAME\_FLAG**){** //flag do nome do ficheiro

packetIndex**++;**

fileName **=** **(**char**\*)** malloc**(**packet**[**packetIndex**]+**1**);**

fileNameLength **=** packet**[**packetIndex**];**

packetIndex**++;**

**for** **(**size\_t i **=** 0**;** i **<** fileNameLength**;** i**++)**

**{**

fileName**[**i**]** **=** packet**[**packetIndex**];**

packetIndex**++;**

**if(**i **==** fileNameLength**-**1**){**

fileName**[**fileNameLength**]** **=** '\0'**;**

**}**

**}**

**}**

**if(**packet**[**packetIndex**]** **==** FILE\_SIZE\_FLAG**){**

packetIndex**+=**2**;**

fileSize **+=** packet**[**packetIndex**]** **<<** 24**;**

packetIndex**++;**

fileSize **+=** packet**[**packetIndex**]** **<<** 16**;**

packetIndex**++;**

fileSize **+=** packet**[**packetIndex**]** **<<** 8**;**

packetIndex**++;**

fileSize **+=** packet**[**packetIndex**];**

**}**

**if(**packetInfo**.**fileSize **!=** fileSize **||** strcmp**(**packetInfo**.**fileName**,**fileName**)!=** 0**){**

printf**(**"Start packet and end packet have different file name and/or different file size\n"**);**

**}**

**}**

int processDataPackets**(**unsigned char**\*** packet**){**

int informationSize **=** 256**\***packet**[**2**]+**packet**[**3**];**

write**(**app**.**fdFile**,**packet**+**4**,**informationSize**);**

**return** 0**;**

**}**

int receiveFile**(){**

unsigned char buffer**[**app**.**packetSize**+**4**];**

int stop **=** 0**;**

int lastSequenceNumber **=** **-**1**;**

int currentSequenceNumber**;**

**while(!**stop**){**

llread**(**app**.**fdPort**,**buffer**);**

**if(**buffer**[**0**]** **==** START\_FLAG**){**

readStartControlPacket**(**buffer**);**

**}**

**else** **if(**buffer**[**0**]** **==** DATA\_FLAG**){**

currentSequenceNumber **=** **(**int**)(**buffer**[**1**]);**

**if(**lastSequenceNumber **>=** currentSequenceNumber **&&** lastSequenceNumber **!=** 254**)**

**continue;**

lastSequenceNumber **=** currentSequenceNumber**;**

processDataPackets**(**buffer**);**

**}**

**else** **if(**buffer**[**0**]** **==** END\_FLAG**){**

readEndControlPacket**(**buffer**);**

stop **=** 1**;**

**}**

**}**

close**(**app**.**fdFile**);**

**return** 0**;**

**}**

### ll.h

#include "utils.h"

#include <stdio.h>

#include <unistd.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

//ConnectionInfo struct setup

speed\_t checkBaudrate**(**char **\*** baudRate**);**

void infoSetup**(**char **\*** baudRate**);**

//Auxiliary functions

int verifyControlByte**(**unsigned char byte**);**

void responseStateMachine**(**enum state**\*** currentState**,** unsigned char byte**,** unsigned char**\*** controlByte**);**

void informationFrameStateMachine**(**enum state**\*** currentState**,** unsigned char byte**,** unsigned char**\*** controlByte**);**

int verifyFrame**(**unsigned char**\*** frame**,**int length**);**

int processControlByte**(**int fd**,** unsigned char **\***controlByte**);**

//Transmitter

void readReceiverResponse**(**int fd**);**

//Receiver

void readTransmitterResponse**(**int fd**);**

int readTransmitterFrame**(**int fd**,** unsigned char **\*** buffer**);**

// ll functions

int llopen**(**char**\*** port**,** int flag**,** char**\*** baudrate**);**

int llwrite**(**int fd**,** unsigned char**\*** packet**,** int length**);**

int llread**(**int fd**,** unsigned char**\*** buf**);**

int llclose**(**int fd**,** int flag**);**

void sigAlarmHandler**();**

void initializeAlarm**();**

void disableAlarm**();**

**typedef** struct **{**

//alarm info

int numTries**;**

int alarmFlag**;**

int ns**;**

speed\_t baudRate**;**

int firstTime**;**

**}** ConnectionInfo**;**

### ll.c

#include "ll.h"

ConnectionInfo info**;**

//struct termios

struct termios oldtio**;**

void sigAlarmHandler**(){**

printf**(**"Timeout!\n"**);**

info**.**alarmFlag **=** 1**;**

info**.**numTries**++;**

**}**

void initializeAlarm**(){**

struct sigaction sa**;**

sa**.**sa\_handler **=** **&**sigAlarmHandler**;**

sigemptyset**(&**sa**.**sa\_mask**);**

sa**.**sa\_flags **=** 0**;**

info**.**alarmFlag **=** 0**;**

sigaction**(**SIGALRM**,** **&**sa**,** **NULL);**

alarm**(**20**);**

**}**

void disableAlarm**(){**

struct sigaction sa**;**

sa**.**sa\_handler **=** **NULL;**

sigaction**(**SIGALRM**,** **&**sa**,** **NULL);**

info**.**alarmFlag **=** 0**;**

alarm**(**0**);**

**}**

speed\_t checkBaudrate**(**char **\*** baudRate**){**

long br **=** strtol**(**baudRate**,** **NULL,** 16**);**

**switch** **(**br**){**

**case** 0xB50**:**

**return** B50**;**

**case** 0xB75**:**

**return** B75**;**

**case** 0xB110**:**

**return** B110**;**

**case** 0xB134**:**

**return** B134**;**

**case** 0xB150**:**

**return** B150**;**

**case** 0xB200**:**

**return** B200**;**

**case** 0xB300**:**

**return** B300**;**

**case** 0xB600**:**

**return** B600**;**

**case** 0xB1200**:**

**return** B1200**;**

**case** 0xB1800**:**

**return** B1800**;**

**case** 0xB2400**:**

**return** B2400**;**

**case** 0xB4800**:**

**return** B4800**;**

**case** 0xB9600**:**

**return** B9600**;**

**case** 0xB19200**:**

**return** B19200**;**

**case** 0xB38400**:**

**return** B38400**;**

**case** 0xB57600**:**

**return** B57600**;**

**case** 0xB115200**:**

**return** B115200**;**

**case** 0xB230400**:**

**return** B230400**;**

**default:**

**return** B38400**;**

**}**

**}**

void infoSetup**(**char **\*** baudRate**){**

info**.**alarmFlag **=** 0**;**

info**.**numTries **=** 0**;**

info**.**ns **=** 0**;**

info**.**baudRate **=** checkBaudrate**(**baudRate**);**

info**.**firstTime **=** 1**;**

**}**

int verifyControlByte**(**unsigned char byte**){**

**return** byte **==** CONTROL\_BYTE\_SET **||** byte **==** CONTROL\_BYTE\_DISC **||** byte **==** CONTROL\_BYTE\_UA **||** byte **==** CONTROL\_BYTE\_RR0 **||** byte **==** CONTROL\_BYTE\_RR1 **||** byte **==** CONTROL\_BYTE\_REJ0 **||** byte **==** CONTROL\_BYTE\_REJ1**;**

**}**

void responseStateMachine**(**enum state**\*** currentState**,** unsigned char byte**,** unsigned char**\*** controlByte**){**

**switch(\***currentState**){**

**case** START**:**

**if(**byte **==** FLAG**){** //flag

**\***currentState **=** FLAG\_RCV**;**

**}**

**break;**

**case** FLAG\_RCV**:**

**if(**byte **==** ADDRESS\_FIELD\_CMD**){** //acknowlegement

**\***currentState **=** A\_RCV**;**

**}**

**else** **if(**byte **!=** FLAG**){**

**\***currentState **=** START**;**

**}**

**break;**

**case** A\_RCV**:**

**if(**verifyControlByte**(**byte**)){**

**\***currentState **=** C\_RCV**;**

**\***controlByte **=** byte**;**

**}**

**else** **if(**byte **==** FLAG**){**

**\***currentState **=** FLAG\_RCV**;**

**}**

**else{**

**\***currentState **=** START**;**

**}**

**break;**

**case** C\_RCV**:**

**if(**byte **==** **(**ADDRESS\_FIELD\_CMD**^(\***controlByte**))){**

**\***currentState **=** BCC\_OK**;**

**}**

**else** **if(**byte **==** FLAG**){**

**\***currentState **=** FLAG\_RCV**;**

**}**

**else{**

**\***currentState **=** START**;**

**}**

**break;**

**case** BCC\_OK**:**

**if(**byte **==** FLAG**){**

**\***currentState **=** STOP**;**

**}**

**else{**

**\***currentState **=** START**;**

**}**

**break;**

**case** STOP**:**

**break;**

**default:**

**break;**

**}**

**}**

void readReceiverResponse**(**int fd**){**

unsigned char byte**;**

enum state state **=** START**;**

unsigned char controlByte**;**

**while(**state **!=** STOP **&&** info**.**alarmFlag **==** 0**){**

**if(**read**(**fd**,&**byte**,**1**)** **<** 0**){**

perror**(**"Error reading byte of the receiver response"**);**

**}**

responseStateMachine**(&**state**,**byte**,&**controlByte**);**

**}**

**}**

void readTransmitterResponse**(**int fd**){**

unsigned char byte**;**

enum state state **=** START**;**

unsigned char controlByte**;**

**while(**state **!=** STOP **&&** info**.**alarmFlag **==** 0**){**

**if(**read**(**fd**,&**byte**,**1**)** **<** 0**){**

perror**(**"Error reading byte of the transmitter response"**);**

**}**

responseStateMachine**(&**state**,**byte**,&**controlByte**);**

**}**

**}**

int llopen**(**char **\***port**,**int flag**,** char**\*** baudrate**){**

struct termios newtio**;**

infoSetup**(**baudrate**);**

printf**(**"BAUD: %d\n"**,** info**.**baudRate**);**

//Open the connection

int fd**;**

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

**if** **(**fd **<**0**)** **{**perror**(**port**);** exit**(-**1**);** **}**

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

perror**(**"tcgetattr"**);**

exit**(-**1**);**

**}**

bzero**(&**newtio**,** **sizeof(**newtio**));**

newtio**.**c\_cflag **=** info**.**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**]** **=** 10**;** /\* inter-unsigned character timer unused \*/

newtio**.**c\_cc**[**VMIN**]** **=** 1**;** /\* blocking read until 5 unsigned chars received \*/

cfsetspeed**(&**newtio**,**info**.**baudRate**);**

tcflush**(**fd**,** TCIOFLUSH**);**

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

perror**(**"tcsetattr"**);**

exit**(-**1**);**

**}**

printf**(**"New termios structure set with baudrate: IN: %d | OUT: %d\n"**,** newtio**.**c\_ispeed**,** newtio**.**c\_ospeed**);**

//First frames' transmission

**if(**flag **==** TRANSMITTER**){**

unsigned char controlFrame**[**5**];**

controlFrame**[**0**]** **=** FLAG**;**

controlFrame**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

controlFrame**[**2**]** **=** CONTROL\_BYTE\_SET**;**

controlFrame**[**3**]** **=** controlFrame**[**1**]** **^** controlFrame**[**2**];**

controlFrame**[**4**]** **=** FLAG**;**

**do{**

write**(**fd**,**controlFrame**,**5**);** //write SET

printf**(**"Sent SET\n"**);**

info**.**alarmFlag **=** 0**;**

initializeAlarm**();**

readReceiverResponse**(**fd**);** //read UA

**if(**info**.**alarmFlag **==** 0**){**

printf**(**"Received UA\n"**);**

**}**

**}** **while(**info**.**numTries **<** MAX\_TRIES **&&** info**.**alarmFlag**);**

disableAlarm**();**

**if(**info**.**numTries **>=** MAX\_TRIES**){**

printf**(**"Exceeded number of maximum tries!\n"**);**

**return** **-**1**;**

**}**

**}**

**else** **if(**flag **==** RECEIVER**){**

readTransmitterResponse**(**fd**);**

printf**(**"Received SET\n"**);**

unsigned char controlFrame**[**5**];**

controlFrame**[**0**]** **=** FLAG**;**

controlFrame**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

controlFrame**[**2**]** **=** CONTROL\_BYTE\_UA**;**

controlFrame**[**3**]** **=** controlFrame**[**1**]** **^** controlFrame**[**2**];**

controlFrame**[**4**]** **=** FLAG**;**

write**(**fd**,**controlFrame**,**5**);** //send UA

printf**(**"Sent UA\n"**);**

**}**

**else{**

printf**(**"Unknown function, must be a TRANSMITTER/RECEIVER\n"**);**

**return** 1**;**

**}**

**return** fd**;**

**}**

int processControlByte**(**int fd**,** unsigned char **\***controlByte**){**

unsigned char byte**;**

enum state state **=** START**;**

**while(**state **!=** STOP **&&** info**.**alarmFlag **==** 0**){**

**if(**read**(**fd**,&**byte**,**1**)** **<** 0**){**

perror**(**"Error reading byte"**);**

**}**

responseStateMachine**(&**state**,**byte**,**controlByte**);**

**}**

**if(\***controlByte **==** CONTROL\_BYTE\_RR0 **&&** info**.**ns **==** 1**){**

printf**(**"Received postive ACK 0\n"**);**

**return** 0**;**

**}**

**else** **if(\***controlByte **==** CONTROL\_BYTE\_RR1 **&&** info**.**ns **==** 0**){**

printf**(**"Received postive ACK 1\n"**);**

**return** 0**;**

**}**

**else** **if(\***controlByte **==** CONTROL\_BYTE\_REJ0 **&&** info**.**ns **==** 1**){**

printf**(**"Received negative ACK 0\n"**);**

**return** **-**1**;**

**}**

**else** **if(\***controlByte **==** CONTROL\_BYTE\_REJ1 **&&** info**.**ns **==** 0**){**

printf**(**"Received negative ACK 1\n"**);**

**return** **-**1**;**

**}**

**else**

**{**

**return** **-**1**;**

**}**

**return** 0**;**

**}**

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

int unsigned charactersWritten **=** 0**;**

unsigned char controlByte**;**

info**.**numTries **=** 0**;**

**if(**info**.**ns **==** 0 **&&** **!**info**.**firstTime**)**

info**.**ns **=** 1**;**

**else** **if(**info**.**ns **==** 1**)**

info**.**ns **=** 0**;**

**do{**

//write frame

unsigned char frameToSend**[**2**\***length**+**7**];**

int frameIndex **=** 4**,** frameLength **=** 0**;**

unsigned char bcc2 **=** 0x00**;**

//Start to make the frame to be sent

frameToSend**[**0**]** **=** FLAG**;** //FLAG

frameToSend**[**1**]** **=** ADDRESS\_FIELD\_CMD**;** //UA

**if(**info**.**ns **==** 0**)**

frameToSend**[**2**]** **=** CONTROL\_BYTE\_0**;**

**else**

**{**

frameToSend**[**2**]** **=** CONTROL\_BYTE\_1**;**

**}**

frameToSend**[**3**]** **=** frameToSend**[**1**]** **^** frameToSend**[**2**];**

**for** **(**size\_t i **=** 0**;** i **<** length**;** i**++){**

bcc2 **^=** buffer**[**i**];**

**if(**buffer**[**i**]** **==** FLAG **||** buffer**[**i**]** **==** ESC\_BYTE**){** //if the byte its equal to the flag or to the escape byte

frameToSend**[**frameIndex**]** **=** ESC\_BYTE**;**

frameIndex**++;**

frameToSend**[**frameIndex**]** **=** buffer**[**i**]** **^** STUFFING\_BYTE**;**

frameIndex**++;**

**}**

**else{**

frameToSend**[**frameIndex**]** **=** buffer**[**i**];**

frameIndex**++;**

**}**

**}**

**if(**bcc2 **==** FLAG **||** bcc2 **==** ESC\_BYTE**){**

frameToSend**[**frameIndex**]** **=** ESC\_BYTE**;**

frameIndex**++;**

frameToSend**[**frameIndex**]** **=** bcc2 **^** STUFFING\_BYTE**;**

frameIndex**++;**

**}**

**else{**

frameToSend**[**frameIndex**]** **=** bcc2**;**

frameIndex**++;**

**}**

frameToSend**[**frameIndex**]** **=** FLAG**;**

frameLength **=** frameIndex**+**1**;**

printf**(**"Before writing to serial...\n"**);**

charactersWritten **=** write**(**fd**,**frameToSend**,**frameLength**);**

printf**(**"Sent frame with sequence number %d\n\n"**,**info**.**ns**);**

initializeAlarm**();**

//read receiver response

**if(**processControlByte**(**fd**,&**controlByte**)** **==** **-**1**){** // if there is an error sending the message, send again

disableAlarm**();**

info**.**alarmFlag **=** 1**;**

**}**

**}while(**info**.**numTries **<** MAX\_TRIES **&&** info**.**alarmFlag**);**

info**.**firstTime **=** 0**;**

disableAlarm**();**

**if(**info**.**numTries **>=** MAX\_TRIES**){**

printf**(**"Exceeded number of maximum tries!\n"**);**

**return** **-**1**;**

**}**

**return** charactersWritten**;**

**}**

void informationFrameStateMachine**(**enum state**\*** currentState**,** unsigned char byte**,** unsigned char**\*** controlByte**){**

**switch(\***currentState**){**

**case** START**:**

**if(**byte **==** FLAG**)** //flag

**\***currentState **=** FLAG\_RCV**;**

**break;**

**case** FLAG\_RCV**:**

**if(**byte **==** ADDRESS\_FIELD\_CMD**)** //acknowlegement

**\***currentState **=** A\_RCV**;**

**else** **if(**byte **!=** FLAG**)**

**\***currentState **=** START**;**

**break;**

**case** A\_RCV**:**

**if(**byte **==** CONTROL\_BYTE\_0 **||** byte **==** CONTROL\_BYTE\_1**){**

**\***currentState **=** C\_RCV**;**

**\***controlByte **=** byte**;**

**}**

**else** **if(**FLAG **==** 0x7E**){**

**\***currentState **=** FLAG\_RCV**;**

**}**

**else{**

**\***currentState **=** START**;**

**}**

**break;**

**case** C\_RCV**:**

**if(**byte **==** **(**ADDRESS\_FIELD\_CMD**^(\***controlByte**)))**

**\***currentState **=** BCC\_OK**;**

**else** **if(**byte **==** FLAG**)**

**\***currentState **=** FLAG\_RCV**;**

**else**

**\***currentState **=** START**;**

**break;**

**case** BCC\_OK**:**

**if(**byte **!=** FLAG**)**

**\***currentState **=** DATA\_RCV**;**

**break;**

**case** DATA\_RCV**:**

**if(**byte **==** FLAG**)**

**\***currentState **=** STOP**;**

**break;**

**case** STOP**:**

**break;**

**}**

**}**

int readTransmitterFrame**(**int fd**,** unsigned char **\*** buffer**){**

int length **=** 0**;**

unsigned char byte**;**

unsigned char controlByte**;**

enum state state **=** START**;**

**while(**state **!=** STOP**){**

**if(**read**(**fd**,&**byte**,**1**)** **<** 0**){**

perror**(**"Error reading byte"**);**

**}**

informationFrameStateMachine**(&**state**,**byte**,&**controlByte**);**

buffer**[**length**]** **=** byte**;**

length**++;**

**}**

**return** length**;**

**}**

int verifyFrame**(**unsigned char**\*** frame**,**int length**){**

unsigned char addressField **=** frame**[**1**];**

unsigned char controlByte **=** frame**[**2**];**

unsigned char bcc1 **=** frame**[**3**];**

unsigned char bcc2 **=** frame**[**length**-**2**];**

unsigned char aux **=** 0x00**;**

//verify if the bcc1 is correct

**if(**controlByte **!=** CONTROL\_BYTE\_0 **&&** controlByte **!=** CONTROL\_BYTE\_1**){**

printf**(**"Error in control byte!\n"**);**

**return** **-**1**;**

**}**

**else** **if(**bcc1 **==** **(**addressField**^**controlByte**)){**

//check bcc2

**for** **(**size\_t i **=** 4**;** i **<** length**-**2**;** i**++)**

**{**

aux**^=**frame**[**i**];**

**}**

**if(**bcc2 **!=** aux**){**

printf**(**"Error in bcc2!\n"**);**

**return** **-**2**;**

**}**

**}**

**return** 0**;**

**}**

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

int received **=** 0**;**

int length **=** 0**;**

unsigned char controlByte**;**

unsigned char auxBuffer**[**131087**];**

int buffIndex **=** 0**;**

info**.**numTries **=** 0**;**

**while(**received **==** 0**){**

length **=** readTransmitterFrame**(**fd**,**auxBuffer**);**

printf**(**"Received frame\n"**);**

**if(**length **>** 0**){**

unsigned char originalFrame**[**2**\***length**+**7**];**

//destuffing

originalFrame**[**0**]** **=** auxBuffer**[**0**];**

originalFrame**[**1**]** **=** auxBuffer**[**1**];**

originalFrame**[**2**]** **=** auxBuffer**[**2**];**

originalFrame**[**3**]** **=** auxBuffer**[**3**];**

int originalFrameIndex **=** 4**;**

int escapeByteFound **=** 0**;**

**for** **(**size\_t i **=** 4**;** i **<** length**-**1**;** i**++)**

**{**

**if(**auxBuffer**[**i**]** **==** ESC\_BYTE**){**

escapeByteFound **=** 1**;**

**continue;**

**}**

**else** **if(**auxBuffer**[**i**]** **==** **(**FLAG**^**STUFFING\_BYTE**)** **&&** escapeByteFound **==** 1**){**

originalFrame**[**originalFrameIndex**]** **=** FLAG**;**

originalFrameIndex**++;**

escapeByteFound **=** 0**;**

**}**

**else** **if(**auxBuffer**[**i**]** **==** **(**ESC\_BYTE**^**STUFFING\_BYTE**)** **&&** escapeByteFound **==** 1**){**

originalFrame**[**originalFrameIndex**]** **=** ESC\_BYTE**;**

originalFrameIndex**++;**

escapeByteFound **=** 0**;**

**}**

**else{**

originalFrame**[**originalFrameIndex**]** **=** auxBuffer**[**i**];**

originalFrameIndex**++;**

**}**

**}**

originalFrame**[**originalFrameIndex**]** **=** auxBuffer**[**length**-**1**];**

controlByte **=** originalFrame**[**2**];**

**if(**verifyFrame**(**originalFrame**,**originalFrameIndex**+**1**)** **!=** 0**){**

**if(**controlByte **==** CONTROL\_BYTE\_0**){**

printf**(**"Frame has 0 as sequence number\n"**);**

unsigned char frameToSend**[**5**];**

frameToSend**[**0**]** **=** FLAG**;**

frameToSend**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

frameToSend**[**2**]** **=** CONTROL\_BYTE\_REJ0**;**

frameToSend**[**3**]** **=** frameToSend**[**1**]** **^** frameToSend**[**2**];**

frameToSend**[**4**]** **=** FLAG**;**

write**(**fd**,**frameToSend**,**5**);**

printf**(**"Sent negative ACK 0\n"**);**

**}**

**else** **if(**controlByte **==** CONTROL\_BYTE\_1**){**

printf**(**"Frame has 1 as sequence number\n"**);**

unsigned char frameToSend**[**5**];**

frameToSend**[**0**]** **=** FLAG**;**

frameToSend**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

frameToSend**[**2**]** **=** CONTROL\_BYTE\_REJ1**;**

frameToSend**[**3**]** **=** frameToSend**[**1**]** **^** frameToSend**[**2**];**

frameToSend**[**4**]** **=** FLAG**;**

write**(**fd**,**frameToSend**,**5**);**

printf**(**"Sent negative ACK 1\n"**);**

**}**

**return** 0**;**

**}**

**else{**

**for** **(**size\_t i **=** 4**;** i **<** originalFrameIndex**-**1**;** i**++)**

**{**

buffer**[**buffIndex**]** **=** originalFrame**[**i**];**

buffIndex**++;**

**}**

**if(**controlByte **==** CONTROL\_BYTE\_0**){**

printf**(**"Frame has 0 as sequence number\n"**);**

unsigned char frameToSend**[**5**];**

frameToSend**[**0**]** **=** FLAG**;**

frameToSend**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

frameToSend**[**2**]** **=** CONTROL\_BYTE\_RR1**;**

frameToSend**[**3**]** **=** frameToSend**[**1**]** **^** frameToSend**[**2**];**

frameToSend**[**4**]** **=** FLAG**;**

write**(**fd**,**frameToSend**,**5**);**

printf**(**"Sent positive ACK 1\n"**);**

**}**

**else** **if(**controlByte **==** CONTROL\_BYTE\_1**){**

printf**(**"Frame has 1 as sequence number\n"**);**

unsigned char frameToSend**[**5**];**

frameToSend**[**0**]** **=** FLAG**;**

frameToSend**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

frameToSend**[**2**]** **=** CONTROL\_BYTE\_RR0**;**

frameToSend**[**3**]** **=** frameToSend**[**1**]** **^** frameToSend**[**2**];**

frameToSend**[**4**]** **=** FLAG**;**

write**(**fd**,**frameToSend**,**5**);**

printf**(**"Sent positive ACK 0\n"**);**

**}**

received **=** 1**;**

**}**

**}**

**}**

**return** buffIndex**;**

**}**

int llclose**(**int fd**,** int flag**){**

//Last frames' transmission

**if(**flag **==** TRANSMITTER**){**

**if(**info**.**numTries **>=** MAX\_TRIES**){**

**return** **-**1**;**

**}**

unsigned char controlFrameDISC**[**5**];**

controlFrameDISC**[**0**]** **=** FLAG**;**

controlFrameDISC**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

controlFrameDISC**[**2**]** **=** CONTROL\_BYTE\_DISC**;**

controlFrameDISC**[**3**]** **=** controlFrameDISC**[**1**]** **^** controlFrameDISC**[**2**];**

controlFrameDISC**[**4**]** **=** FLAG**;**

**do{**

write**(**fd**,**controlFrameDISC**,**5**);** //write DISC

printf**(**"Sent DISC\n"**);**

info**.**alarmFlag **=** 0**;**

initializeAlarm**();**

readReceiverResponse**(**fd**);** //read DISC

**}** **while(**info**.**numTries **<** MAX\_TRIES **&&** info**.**alarmFlag**);**

**if(**info**.**alarmFlag **==** 0**){**

printf**(**"Received DISC\n"**);**

**}**

disableAlarm**();**

**if(**info**.**numTries **>=** MAX\_TRIES**){**

printf**(**"Exceeded number of maximum tries!\n"**);**

**return** **-**1**;**

**}**

**else{**

unsigned char controlFrameUA**[**5**];**

controlFrameUA**[**0**]** **=** FLAG**;**

controlFrameUA**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

controlFrameUA**[**2**]** **=** CONTROL\_BYTE\_UA**;**

controlFrameUA**[**3**]** **=** controlFrameUA**[**1**]** **^** controlFrameUA**[**2**];**

controlFrameUA**[**4**]** **=** FLAG**;**

write**(**fd**,**controlFrameUA**,**5**);** //write UA after receiving DISC

printf**(**"Sent UA\n"**);**

sleep**(**1**);**

**}**

**}**

**else** **if(**flag **==** RECEIVER**){**

**if(**info**.**numTries **>=** MAX\_TRIES**){**

**return** **-**1**;**

**}**

readTransmitterResponse**(**fd**);**

printf**(**"Receveid DISC\n"**);**

unsigned char controlFrame**[**5**];**

controlFrame**[**0**]** **=** FLAG**;**

controlFrame**[**1**]** **=** ADDRESS\_FIELD\_CMD**;**

controlFrame**[**2**]** **=** CONTROL\_BYTE\_DISC**;**

controlFrame**[**3**]** **=** controlFrame**[**1**]** **^** controlFrame**[**2**];**

controlFrame**[**4**]** **=** FLAG**;**

write**(**fd**,**controlFrame**,**5**);** //send DISC

printf**(**"Sent DISC\n"**);**

readTransmitterResponse**(**fd**);**

printf**(**"Received UA\n"**);**

**}**

**else{**

printf**(**"Unknown function, must be a TRANSMITTER/RECEIVER\n"**);**

**return** 1**;**

**}**

//Close the connection

tcflush**(**fd**,** TCIOFLUSH**);**

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

perror**(**"tcsetattr"**);**

exit**(-**1**);**

**}**

close**(**fd**);**

**return** 0**;**

**}**

### utils.h

#include <termios.h>

#define BAUDRATE B38400

#define MODEMDEVICE "/dev/ttyS1"

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

#define TRANSMITTER 0

#define RECEIVER 1

#define FLAG 0x7E

#define ADDRESS\_FIELD\_CMD 0x03

#define CONTROL\_BYTE\_0 0x00

#define CONTROL\_BYTE\_1 0x40

#define CONTROL\_BYTE\_SET 0x03

#define CONTROL\_BYTE\_DISC 0x0B

#define CONTROL\_BYTE\_UA 0x07

#define CONTROL\_BYTE\_RR0 0x05

#define CONTROL\_BYTE\_RR1 0x85

#define CONTROL\_BYTE\_REJ0 0x01

#define CONTROL\_BYTE\_REJ1 0x81

#define ESC\_BYTE 0x7D

#define STUFFING\_BYTE 0x20

#define START\_FLAG 0x02

#define END\_FLAG 0x03

#define FILE\_NAME\_FLAG 0x01

#define FILE\_SIZE\_FLAG 0x00

#define DATA\_FLAG 0x01

#define BYTE\_MASK 0xFF

#define MAX\_TRIES 3

enum state**{**START**,** FLAG\_RCV**,** A\_RCV**,** C\_RCV**,** BCC\_OK**,** STOP**,** DATA\_RCV**};**

### proj.c

#include "application.h"

#include <sys/time.h>

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

**{**

**if(**argc **<** 6**){**

printf**(**"Usage: serialPort flag file packetSize baudrate\n"**);**

**return** **-**1**;**

**}**

**if** **(**

**((**strcmp**(**"/dev/ttyS0"**,** argv**[**1**])!=**0**)** **&&**

**(**strcmp**(**"/dev/ttyS1"**,** argv**[**1**])!=**0**)** **&&**

**(**strcmp**(**"/dev/ttyS10"**,** argv**[**1**])!=**0**)** **&&**

**(**strcmp**(**"/dev/ttyS11"**,** argv**[**1**])!=**0**)))** **{**

printf**(**"Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n"**);**

exit**(**1**);**

**}**

int fd**;**

int flag**;**

int packetSize **=** atoi**(**argv**[**4**]);**

**if(**strcmp**(**"1"**,**argv**[**2**])** **==** 0**){**

flag **=** RECEIVER**;**

**}**

**else** **if(**strcmp**(**"0"**,**argv**[**2**])** **==** 0**){**

flag **=** TRANSMITTER**;**

**}**

**else{**

printf**(**"Invalid argument for flag\n"**);**

exit**(**2**);**

**}**

**if(**strcmp**(**argv**[**5**],**"B50"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B75"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B110"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B134"**)** **!=** 0 **&&**

strcmp**(**argv**[**5**],**"B150"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B200"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B300"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B600"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B1200"**)** **!=** 0 **&&**

strcmp**(**argv**[**5**],**"B1800"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B2400"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B4800"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B9600"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B19200"**)** **!=** 0 **&&**

strcmp**(**argv**[**5**],**"B38400"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B57600"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B115200"**)** **!=** 0 **&&** strcmp**(**argv**[**5**],**"B230400"**)** **!=** 0**)**

**{**

printf**(**"Baurate must be one of the following values: B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200, B38400, B57600, B115200, B230400\n"**);**

**return** **-**2**;**

**}**

struct timeval start**,** end**;**

double time\_taken**;**

gettimeofday**(&**start**,** **NULL);**

fd **=** llopen**(**argv**[**1**],**flag**,** argv**[**5**]);**

applicationSetUp**(**argv**[**3**],**packetSize**,**fd**);**

**if(**fd **>** 0**){**

**if(**flag **==** RECEIVER**){**

receiveFile**();**

**}**

**else** **if(**flag **==** TRANSMITTER**){**

sendFile**();**

**}**

llclose**(**fd**,**flag**);**

**}**

gettimeofday**(&**end**,** **NULL);**

time\_taken **=** end**.**tv\_sec **+** end**.**tv\_usec **/** 1e6 **-**

start**.**tv\_sec **-** start**.**tv\_usec **/** 1e6**;** // in seconds

printf**(**"Time spent: %lf\n"**,**time\_taken**);**

**return** 0**;**

**}**

## Anexo II – Resultados em Ambiente de Laboratório

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tamanho dos pacotes (bytes)** | **Tempo(segundos)** | **Bit Rate(bits/segundo)** | **Eficiência** | **Eficiência Média** |
| 16 | 34,445066 | 43236,38108 | 0,375315808 | 0,3764666 |
| 34,294896 | 43425,70393 | 0,376959236 |
| 34,279844 | 43444,7718 | 0,377124755 |
| 32 | 25,718336 | 57907,3234 | 0,502667738 | 0,502729429 |
| 25,711655 | 57922,37022 | 0,502798353 |
| 25,71555 | 57913,59703 | 0,502722196 |
| 64 | 21,513826 | 69224,32114 | 0,600905565 | 0,600764748 |
| 21,511072 | 69233,18373 | 0,600982498 |
| 21,53172 | 69166,79206 | 0,600406181 |
| 128 | 19,368117 | 76893,38101 | 0,667477266 | 0,66744755 |
| 19,366155 | 76901,17114 | 0,667544888 |
| 19,372667 | 76875,3213 | 0,667320497 |
| 256 | 18,291416 | 81419,61235 | 0,706767468 | 0,706786067 |
| 18,290829 | 81422,22531 | 0,70679015 |
| 18,290559 | 81423,42724 | 0,706800584 |
| 512 | 17,748392 | 83910,70019 | 0,728391495 | 0,728288903 |
| 17,752556 | 83891,01828 | 0,728220645 |
| 17,751729 | 83894,92652 | 0,72825457 |
| 1024 | 17,481011 | 85194,15725 | 0,739532615 | 0,739511393 |
| 17,481644 | 85191,07242 | 0,739505837 |
| 17,481883 | 85189,90775 | 0,739495727 |
| 2048 | 17,347021 | 85852,20483 | 0,745244834 | 0,745221035 |
| 17,347155 | 85851,54165 | 0,745239077 |
| 17,348549 | 85844,64326 | 0,745179195 |
| 4096 | 17,281079 | 86179,80393 | 0,748088576 | 0,748086904 |
| 17,282242 | 86174,00451 | 0,748038234 |
| 17,280032 | 86185,02558 | 0,748133903 |
| 8192 | 17,245917 | 86355,51244 | 0,749613823 | 0,749612418 |
| 17,245835 | 86355,92304 | 0,749617387 |
| 17,246096 | 86354,61614 | 0,749606043 |
| 16384 | 17,229597 | 86437,30901 | 0,750323863 | 0,750318289 |
| 17,229337 | 86438,6134 | 0,750335186 |
| 17,230241 | 86434,07832 | 0,750295819 |
| 32768 | 17,220743 | 86481,75053 | 0,75070964 | 0,750709117 |
| 17,220852 | 86481,20314 | 0,750704888 |
| 17,22067 | 86482,11713 | 0,750712822 |

|  |  |
| --- | --- |
| **Baudrate** | 115200 |
| **Tamanho do ficheiro (bytes)** | 186160 |