

Redes de Computadores

### RCOM – 3MIEIC06

1º Trabalho Laboratorial – Ligação de Dados

João Álvaro Cardoso Soares Ferreira – [up201605592@fe.up.pt](mailto:up201605592@fe.up.pt)

[João Augusto dos Santos Lima](https://sigarra.up.pt/feup/pt/fest_geral.cursos_list?pv_num_unico=201605314) – [up201605314@fe.up.pt](mailto:up201605314@fe.up.pt)

Índice

[RCOM – 3MIEIC06 1](#_Toc529741942)

[Sumário 3](#_Toc529741943)

[Introdução 3](#_Toc529741944)

[Arquitetura 4](#_Toc529741945)

[Estrutura do Código 4](#_Toc529741946)

[Camada de Ligação de Dados 4](#_Toc529741947)

[Camada de Aplicação 6](#_Toc529741948)

[Casos de Uso 6](#_Toc529741949)

[Protocolo de Ligação de Dados 6](#_Toc529741950)

[Implementação 6](#_Toc529741951)

[Protocolo de Aplicação de Dados 7](#_Toc529741952)

[Validação 9](#_Toc529741953)

[Eficiência do Protocolo de Ligação de Dados 9](#_Toc529741954)

[Comparação a Stop & Wait 9](#_Toc529741955)

[Conclusão 9](#_Toc529741956)

[Anexo I 10](#_Toc529741957)

[Connect.c 10](#_Toc529741958)

[Data.c 13](#_Toc529741959)

[Disconnect.c 24](#_Toc529741960)

[Reader.c 27](#_Toc529741961)

[Shared.c 28](#_Toc529741962)

[StateMachine.c 29](#_Toc529741963)

[TestApplication.c 34](#_Toc529741964)

[Writer.c 40](#_Toc529741965)

## Sumário

Neste trabalho, o nosso objetivo foi aplicar os conceitos lecionados na unidade curricular de Rede de Computadores relativamente à transferência de informação de forma prática. Para isso foi importante dominar os conceitos e utilização de diferentes camadas, transferência de informação via serial port, manipulação de dados para gerar verificações, entre outros.

Tendo o trabalho terminado, podemos concluir que o trabalho foi realizado como esperado, tendo existido algumas dificuldades intermédias mas

## Introdução

O objetivo do primeiro trabalho laboratorial foi implementar um protocolo de ligação de dados e utilizá-lo para transferir ficheiros entre dois computadores através do serial port, de acordo com as especificações definidas para tal no guião fornecido.

As especificações mencionadas implicaram a utilização de protocolos de encapsulamento, deteção e correção de erros, sincronização entre os computadores através da confirmação da receção das tramas encapsuladas (algo que foi efetuado servindo-se do uso de máquinas de estado) e abertura/fecho da ligação.

O relatório, que tem como objetivo documentar o trabalho realizado, está estruturado da seguinte forma:

* **Arquitetura** – Explicação de como cada secção do trabalho funciona e interage com as restantes secções.
* **Estrutura do Código** – Exibição das principais estruturas de dados, principais funções e APIs.
* **Casos de uso principais** – Identificação destes e apresentação da sequência de chamada de funções.
* **Protocolo de ligação lógica** – Especificação dos principais aspetos funcionais para a ligação lógica, descrição da estratégia da implementação destes aspetos e apresentação de extratos de código.
* **Protocolo de aplicação** - Especificação dos principais aspetos funcionais da aplicação, descrição da estratégia da implementação destes aspetos e apresentação de extratos de código.
* **Validação** - Descrição dos testes efetuados com apresentação quantificada dos resultados.
* **Eficiência do protocolo de ligação de dados** – Especificação da eficiência do protocolo desenvolvido, com recurso a medições feitas durante a execução. Comparação ao protocolo Stop&Wait e descrição deste método.
* **Conclusões** – Resumo da informação do relatório e reflexão sobre o alcance dos objetivos.

## Arquitetura

O nosso projeto está separado em camadas independentes, de acordo com o princípio de independência de camadas que nos era especificado – a camada de ligação de dados e a camada de aplicação de dados.

De modo a simplificar o código e o tornar mais organizado, separamos também as funções da camada de ligação de dados em vários ficheiros: o estabelecimento (connect.c) e fecho de ligação (disconnect.c) (que sincronizam entre si a ligação com um mecanismo de alarme), a gestão dos estados para verificação e teste do envio de dados noutro (stateMachine.c) e, num quarto, a implementação do mecanismo de stuffing e envio de dados (data.c). Existe também um último ficheiro (shared.c) que permite a selação de escrita ou leitura e auxilia o connect.c e o disconnect.c no envio de flags.

Na camada de aplicação, que foi implementada num só ficheiro – testApplication.c - contém a criação e envio dos pacotes é feito e é mostrada uma interface que dá informação ao utilizador sobre a transferência do ficheiro.

Para além disto, separamos a interface do utilizador também em dois ficheiros, reader.c e writer.c, cada um com a sua respetiva função main, que inicializa as restantes camadas.

## Estrutura do Código

### Camada de Ligação de Dados

#### Shared

void setRead(); - Guarda a máquina como sendo do tipo emissor.

void setWrite(); - Guarda a máquina como sendo do tipo recetor.

int getType(); - Retorna o número associado ao tipo da máquina.

int writeByte(int fd,unsigned char byte); - Envia um byte para a outra máquina

void sendMessage( char \*type, char \* typeSender,int fd); - Determina a flag a escreve-a para writeByte.

#### Connect

int llOpen( char \*canal); - Abre a conexão entre o emissor e o recetor. É necessário que a função setRead ou setWrite seja chamada antes para identificar o tipo.

void makeConnectionReceiver(int fd); - Estabelece a conexão, feita por parte do recetor.

void makeConnectionSender(int fd); - Estabelece a conexão, feita por parte do emissor.

void touchConnectSender(); - Função chamada quando o sinal do relógio é recebido, só chamada no início do programa.

void closeL(int fd); - Aplica a estrutura termios, função auxiliar a llClose.

#### 

#### Disconnect

int llClose(int fd); - Fecha a ligação, deve ser chamado setWrite ou setRead antes de modo a determinar se a máquina em questão é emissor ou recetor.

void disconnectReceiver(int fd); - Faz a desconexão por parte do recetor.

void disconnectSender(int fd); - Faz a disconexão por parte do emissor.

#### StateMachine

int getStateSet();

int getStateUa();

int getStateDisc();

Devolvem os states das state machines

void resetStates();

Modificam os valores das state machines para os valores iniciais.

void stateMachineSET(unsigned char message);

void stateMachineUA(unsigned char message);

void stateMachineDisc(unsigned char message);

void stateMachineDisc2(unsigned char message);

void stateMachineUaDisc(unsigned char message);

As state machines que usamos para sincronizar as máquinas, tanto no estabelecimento e restabelecimento de conexões como na sincronização dos dados e verificação de erros.

#### Data

int llwrite(int fd,unsigned char \*data,int length); - Escreve a trama de informação no descritor, esperando por uma resposta do recetor e reenviando no caso de timeout.

int llRead(int fd,unsigned char \*guardar2); - Lê informação e envia uma mensagem de retorno de acordo com esta ser apropriada ou não (se for manda Ready; se não for, manda REJ).

void writeStuff(unsigned char data,int fd); - Recebe um byte unsigned char e transforma-o caso seja necessário fazendo stuffing.

void touch2(); - Função que lida com o alarme, apenas chamada durante transferência de dados.

unsigned char unStuff(unsigned char first,unsigned char second); - Verifica se um byte sofre stuffing e retorna-o (com o de-stuffing efetuado).

void sendDataAux(unsigned char \*data,int length,int fd); - Envia a trama com os campos de controlo.

int comparer(unsigned char \*n1,int lenght); - Compara um array com a ultima trama recebida; caso não haja uma trama recebida ainda, o valor dessa trama fica como última.

void changestate2Read(unsigned char message, unsigned char bcc);

void changestate2Write(unsigned char message);

void changestate2WriteREJ(unsigned char message);

Maquinas de estados para as mensagens Ready e REJ.

### Camada de Aplicação

#### testApplication

void applicationSend(int fd, char\* path); - Abre e interpeta o ficheiro a enviar, sevindo-se de sendData para enviar os pacotes de dados e de sendControl para enviar os pacotes de controlo.

void sendControl(int fd, int size, char\* path, int startOrEnd); - Envia pacotes de control

void sendData(unsigned char \*text, int fd,int seq,int leng) - Envia pacotes de dados

void receiveData( int fd); - Recebe todos os dados interpreta-os servindo-se de receiveDataRead e receiveControl, criando um ficheiro através da informação recebida.

void receiveDataRead(int fd) – Interpreta pacotes de dados

void receiveControl(int fd, int startOrEnd) – Interpreta pacotes de controlo

## Casos de Uso

Tendo duas máquinas preparadas para a execução do programa, determina-se qual é o emissor e o recetor e compila-se e chama-se a função main respetiva: writer para o emissor e reader para o recetor. Para o reader, o parâmetro basta ser o port utilizado: “/dev/ttyS0” enquanto que o writer deve ter esse mesmo port e o nome do ficheiro para ser enviado.

Após ambas as máquinas terem iniciado a execução do programa, é apresentada uma barra de progresso, uma percentagem e tanto o tamanho do ficheiro total como a quantidade de dados já transferida. Ocorrendo alguma interferência ou paragem, é reconhecido o timeout pela interface que informa o utilizador do que o que está a ocorrer, seguindo-se da continuação do reenvio do ficheiro ou do fecho do programa.

Uma vez que esteja o ficheiro enviado completamente, o programa informa o utilizador e encerra.

## Protocolo de Ligação de Dados

O objetivo deste protocolo é o envio de dados entre as duas máquinas, realizando sincronização entre ambas e verificação de erros de modo a que a informação seja passada corretamente. Como já foi mencionado, esta foi implementada como uma só camada mas com as suas funções principais divididas em vários ficheiros, para que a organização do código seja mais simples. Estes aspetos essenciais são:

* Configurar a porta série
* Utilizar a porta série para estabelecer uma ligação entre duas máquinas
* Envio e receção de mensagens, comandos e ficheiros através da porta série
* Stuffing e destuffing de bytes
* Controlo e recuperação de erros durante a transferência.

### Implementação

A configuração da porta série é feita na função **llOpen** que altura as configurações para as pretendidas. Esta função também chama as funções **makeConnectionSender** e **makeConnectionReceiver**, dependendo de se a máquina é o emissor ou o recetor. Estas funções enviam tramas SET e UA através de **WriteByte**, respetivamente, e aguardam a resposta da outra máquina para poder estabelecer a ligação (sendo UA a resposta a SET). O alarme é assim ativado e pode a partir daqui ser desencadeado, contando como timeout, até chegar ao possível máximo.

O fecho da ligação é feito por **llClose** que, tal como **llOpen**, chama uma versão diferente da função de fecho para o recetor e para o emissor – **disconnectReceiver** e **disconnectSender** que enviam tramas DISC apropriadas e aguardam uma resposta com as máquinas de estado **stateMachineDisc** e **stateMachineDisc2**, respondendo com uma do mesmo tipo. Após isto, o emissor envia uma mensagem UA ao recetor que este recebe, fechando assim o programa.

A função principal para a escrita de dados é **llWrite** que tem **sendDataAux** como auxiliar e recebe os dados, o descritor e o tamanho dos dados como parâmetros. A primeira chama a segunda e escreve a trama de informação no descritor, aguarda a resposta e lida com timeouts e erros (reenviando até três vezes), enquanto que a segunda escreve e envia a trama com os campos de controlo e os dados em si, o que é feito chamando **writeStuff**, que efetua o stuffing. Foi feita a separação em duas funções de modo a não tornar o código de uma só função demasiado extensa.

As respostas do recetor a **llWrite** podem ser do tipo RR/Ready, indicando o sucesso na transmissão de informação, ou REJ, indicando erro. No caso de REJ ou de não obter resposta, a trama é reenviada como já foi mencionado. As verificações das mensagens são feitas com as funções **changestate2Write** e **changestate2WriteREJ.**

O stuffing tem o seguinte formato, em que F é a flag de início de trama, A é o campo de endereço, C é campo de controlo, D são os campos de informação e os BCC são campos de proteção independentes:

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Figura 1 - Formatação do Stuffing

O de-stuffing é feito em **unStuff**, que é chamado por **llRead**, a função responsável pela leitura da trama enviada pelo emissor. Esta é verificada com uma máquina de estados que verifica cada byte da trama, voltando ao início em caso de obter um valor incorreto. Esta verifica também, dando uso a **compare**, se o byte enviado é duplicado, descartando-o e enviando RR para o emissor. Como já foi mencionado, no caso de leitura com sucesso envia RR ao emissor, no caso de erro de leitura envia um REJ. As verificações são efetuadas através de máquinas de estados, nomeadamente **changestate2Read.**

## Protocolo de Aplicação de Dados

A camada de aplicação de dados está principalmente encarregue de gerar, enviar e interpretar pacotes de dados e controlo. Estes pacotes são depois divididos em tramas pela camada de ligação de dados, embora isto em nada afete o funcionamento da camada de aplicação de dados. Os dados são obtidos a partir de um ficheiro indicado pelo utilizador, pelo que esta mesma camada tem também de ser capaz de ler desse ficheiro e criar um novo com os dados transferidos, após a receção.

Quando executado pelo emissor, o programa indica que é o emissor, executa **llOpen** e chama **applicationSend**, tendo esta como argumentos o descritor e o nome do ficheiro, cuja localização é assumida estar na mesma pasta do código fonte.

Na execução de **applicationSend** esta abre e lê o ficheiro, determinando o seu nome e tamanho. **sendControl** é chamado com esta informação como parâmetros e com a indicação de que é um pacote de controlo inicial, sendo que esta função elabora um pacote de controlo com base no nome e tamanho do ficheiro, codificada em TLV (type, length e value). Esta depois é enviada através de **llWrite** com o seguinte formato:

Uma imagem com objeto

Descrição gerada automaticamente

Figura 2 - Formato de Control Package

Em que C é a flag para um pacote de controlo Start (2) e a informação codificada em TLV1 e TLV2 é o tamanho e nome do ficheiro, respetivamente.

Após isto, **applicationSend** divide o conteúdo do ficheiro em porções de, no máximo, 128 bytes, e chama a função **sendData**  que os envia para o recetor em pacotes com o seguinte formato:



Figura 3 - Formato de Data Package

Em que C é a flag que indica um pacote de dados (1), N o número de sequência, L2 e L1 indicam o número de octetos do campo de dados e, por fim, P1-Pk é o campo de dados em si. Enquanto isto, **applicationSend** informa o utilizador do progresso do ficheiro.

Por fim, do mesmo modo que enviou o pacote de controlo inicial, **applicationSend** envia um pacote de controlo final (ou seja, igual, mas com a flag de End (3)).

Enquanto isto, o recetor terá executado o programa que, tal como o emissor, o identificou e executou **llOpen**, procedendo a executar a função **receiveData** que toma como parâmetro o descritor. Esta função chama as funções **receiveControl** e **receiveDataRead** para interpretar os pacotes de dados e criar o ficheiro novo na máquina do recetor, sendo que estas funções por sua vez servem-se de **llRead** para ler a informação enviada pelo emissor.

Após a receção do pacote de controlo final, ambas as máquinas encerram a camada de aplicação.

## Validação

De modo a verificar o bom funcionamento do programa desenvolvido, efetuamos os seguintes testes:

* Envio de ficheiros de vários tipos (txt, png, mp3, mp4, gif).
* Envio de ficheiros com diferentes dimensões (de 4KB a 20MB).
* Interferência durante o envio através do fio de cobre.
* Interferência através de interrupções prolongadas da ligação com o botão.
* Retirar e voltar a colocar o cabo da porta série durante a execução.
* Diferentes valores para tamanho de pacotes e BAUDRATE

Todos estes testes foram passados com sucesso.

## Eficiência do Protocolo de Ligação de Dados

### Comparação a Stop & Wait

O protocolo *Stop & Wait* consiste no envio de uma resposta de confirmação pelo recetor aquando de qualquer transmissão de dados por parte do emissor. Caso a receção tenha sido bem-sucedida o recetor responde com ACK (acknowledgement) e caso tenha sido mal sucedida o recetor responde com NACK. Nos casos de a resposta ser NACK ou não existir resposta ao fim de alguns segundos pré-determinados, o emissor envia os dados novamente.

O protocolo de ligação de dados que desenvolvemos é em muito semelhante ao protocolo *Stop & Wait* já que, tal como este, o recetor aguarda pela resposta do emissor e gera um timeout caso esta não exista. As respostas do emissor são também estas idênticas às de *Stop & Wait*, com a adição dos pormenores de que tomamos em conta se a trama enviada é Ns ou Nr, de modo a dar uma resposta apropriada (REJ0 e REJ1 em vez de NACK) e também verificamos a possibilidade de uma trama ser repetida à anterior de modo a poder corrigir esse erro.

## Conclusão

O desenvolvimento deste projeto levou a uma enorme expansão do conhecimento dos membros do nosso grupo sobre os temas abordados, nomeadamente a comunicação de dados entre dois sistemas e os protocolos para isto utilizados, a comunicação via porta série e o conceito de independência de camadas.

Tendo em conta os nossos resultados, podemos afirmar que os objetivos foram cumpridos já que fomos capazes de aplicar estes conceitos com sucesso no nosso projeto – tornando as camadas completamente independentes, fazendo uso dos protocolos de modo a não ter erros na passagem dos ficheiros e, por último, a assegurar que as máquinas se mantêm sincronizadas.

# Anexo I

### Connect.c

#include "connect.h"

#include "stateMachine.h"

struct termios \*oldtio,\*newtio;

int connect\_count=1;

int connect\_fd=0;

int connect\_alarm=0;

/\*\*

\* @brief Funcao que e chamada quando o sinal do relogio e recebido. Só é chamada quando o programa esta a tentar estabelecer a conecao.

\*

\*/

void touchConnectSender()

{

connect\_alarm=1;

printf("Timeout number %d in SET message response. Resending message...\n", connect\_count);

connect\_count++;

if(connect\_count>=4)

{

printf("Couldn't make connection. Exiting...\n");

exit(1);

}

}

/\*\*

\* @brief Estabelece a conecao. Deve ser feita por parte do Recetor

\*

\* @param fd Descritor do ficheiro

\*/

void makeConnectionReceiver(int fd)

{

int res;

unsigned char receive;

connect\_fd=fd;

connect\_count=0;

//esperar pela mensagem de set do emissor

printf("Waiting for the SET message from Sender...\n");

while(getStateSet()!=5){

res=read(fd,&receive,1);

if(res==0)continue;

stateMachineSET(receive);

}

//Mandar a mensagem de UA

printf("SET message received. Sending UA message to Receiver...\n");

sendMessage("UA","R",fd);

printf("UA message sent to Receiver.\n");

}

/\*\*

\* @brief Estabelece a conecao. deve ser feita por parte do Emissor

\*

\* @param fd Descritor do ficheiro

\*/

void makeConnectionSender(int fd)

{

signal(SIGALRM, touchConnectSender);

int res;

unsigned char receive;

connect\_fd=fd;

connect\_count=0;

//Mandar mensagem de SET

printf("Sending SET message to Receiver...\n");

sendMessage("SET","W",fd);

printf("SET message sent to Receiver. Waiting for response...\n");

//Esperar pela resposta do recetor

alarm(WAITTIME);

while(getStateUa()!=5){

res=read(fd,&receive,1);

if(connect\_alarm)

{

signal(SIGALRM, touchConnectSender);

sendMessage("SET","W",fd);

connect\_alarm=0;

resetStates();

alarm(WAITTIME);

continue;

}

if(res==0) continue;

stateMachineUA(receive);

}

connect\_alarm=0;

alarm(0);

printf("UA message received from Receiver.\n");

}

/\*\*

\* @brief Abre a conecao entre o emissor e o recetor. É necessario que a funcao setRead ou setWrite seja chamada antes para identificar o tipo.

\*

\* @param canal Path com o canal a usar. Normal é o /dev/ttyS0

\* @return int Retorna o descrito do ficheiro correspondente ao canal

\*/

int llOpen( char \*canal)

{

int fd;

oldtio=(struct termios \*)malloc(sizeof(struct termios));

newtio=(struct termios \*)malloc(sizeof(struct termios));

// Open port to read

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

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

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

perror("tcgetattr");

exit(-1);

}

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

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

(\*newtio).c\_iflag = IGNPAR;

(\*newtio).c\_oflag = 0;

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

(\*newtio).c\_lflag = 0;

(\*newtio).c\_cc[VTIME] = 0;

(\*newtio).c\_cc[VMIN] = 1;

tcflush(fd, TCIOFLUSH);

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

perror("tcsetattr");

exit(-1);

}

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

if(getType())

{

makeConnectionReceiver(fd);

}

else

{

makeConnectionSender(fd);

}

return fd;

}

/\*\*

\* @brief Aplica a estrutura termios antiga. Função auxiliar do llClose

\*

\* @param fd Descritor do ficheiro do canal

\*/

void closeL(int fd)

{

tcsetattr(fd,TCSANOW,oldtio);

}

### Data.c

#include "data.h"

#include "stateMachine.h"

void changestate2Read(unsigned char message, unsigned char bcc);

void changestate2Write(unsigned char message);

void changestate2WriteREJ(unsigned char message);

int s = 0;

int state2 = 0;

int state3 = 0;

unsigned char last\_tram[10 \* PackageSize];

int ja\_last=0;

int count2 = 1;

int data\_alarm = 0;

/\*\*

\* @brief Funcao que trata do alarm quando e recebido este sinal. Esta função é só chamada quando estão a ser transferidos dados

\*

\*/

void touch2()

{

data\_alarm = 1;

printf("Timeout number %d in Data message response. Resending data...\n", count2);

count2++;

if (count2 >= 4)

{

printf("Couldn't send data. Exiting...\n");

exit(1);

}

}

/\*\*

\* @brief Compara um array de unsigned com o ultimo tram recebido. Caso ainda não haja um ultimo tram, este passa a ser esse

\*

\* @param n1 Um array de unsigned chars

\* @param lenght Tamanho do array

\* @return int retorna 1 caso seja diferente ou 0 caso seja igual

\*/

int comparer(unsigned char \*n1,int lenght)

{

if(ja\_last)

{

int igual=0;

for(int i=0;i<lenght;i++)

{

if(n1[i]!=last\_tram[i])

{

igual=1;

}

last\_tram[i]=n1[i];

}

return igual;

}

else

{

ja\_last=1;

for(int i=0;i<lenght;i++)

{

last\_tram[i]=n1[i];

}

return 1;

}

}

/\*\*

\* @brief Recebe um byte de unsigned char e tranforma-o caso seja necessario. Stuffing

\*

\* @param data Byte que vai sofrer stuffing

\* @param fd Descritor do ficheiro

\*/

void writeStuff(unsigned char data, int fd)

{

unsigned char send;

if (data == FLAG)

{

send = ESCAPE;

writeByte(fd, send);

send = ESCAPEF;

writeByte(fd, send);

}

else if (data == ESCAPE)

{

send = ESCAPE;

writeByte(fd, send);

send = ESCAPEE;

writeByte(fd, send);

}

else

{

send = data;

writeByte(fd, send);

}

}

/\*\*

\* @brief Verifica se um byte sofrer stuffing e retorna este byte

\*

\* @param first Primeiro byte a verificar

\* @param second Segundo byte a verificar

\* @return unsigned char Retorna 0 caso não haja stuffing ou diferente caso haja stuffing

\*/

unsigned char unStuff(unsigned char first, unsigned char second)

{

if (first == ESCAPE && second == ESCAPEF)

{

return 0x7e;

}

else if (first == ESCAPE && second == ESCAPEE)

{

return ESCAPE;

}

else

return 0x00;

}

/\*\*

\* @brief Manda a trama com os campos de controlos

\*

\* @param data Informação que eu quero enviar

\* @param length Tamnho da trama a enviar

\* @param fd Descritor do ficheiro

\*/

void sendDataAux(unsigned char \*data, int length, int fd)

{

unsigned char bcc1 = 0, bcc2 = 0;

unsigned char send;

bcc1 ^= Aemi;

if (s == 0)

bcc1 ^= S0;

else

bcc1 ^= S1;

for (int i = 0; i < length; i++)

{

bcc2 ^= data[i];

}

send = FLAG;

writeByte(fd, send);

send = Aemi;

writeByte(fd, send);

if (s == 0)

{

send = S0;

}

else

{

send = S1;

}

writeByte(fd, send);

writeStuff(bcc1, fd);

for (int i = 0; i < length; i++)

{

writeStuff(data[i], fd);

}

writeStuff(bcc2, fd);

send = FLAG;

writeByte(fd, send);

}

/\*\*

\* @brief Escreve a trama de informacao no descritor. Espera por uma resposta por parte do recetor. Caso haja timeout ou a mensagem seja mal recebida volta a enviar

\*

\* @param fd Descritor do ficheiro

\* @param data Informacao que eu pretendo enviar

\* @param length Comprimento da informacao

\* @return int Comprimento enviado. Deve ser sempre length, ou o programa termina a meio

\*/

int llwrite(int fd, unsigned char \*data, int length)

{

int res;

unsigned char receive;

data\_alarm = 0;

state2 = 0;

state3 = 0;

sendDataAux(data, length, fd);

signal(SIGALRM, touch2);

printf("Data sent. Waiting for the response...\n");

alarm(WAITTIME);

while (state2 != 5)

{

signal(SIGALRM, touch2);

res = read(fd, &receive, 1);

if (data\_alarm)

{

data\_alarm = 0;

sendDataAux(data, length, fd);

state2 = 0;

state3 = 0;

resetStates();

alarm(WAITTIME);

continue;

}

if (res == 0)

continue;

changestate2Write(receive);

changestate2WriteREJ(receive);

if (state3 == 5)

{

printf("REJ message received ...\n");

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

sendDataAux(data, length, fd);

printf("Data resent.\n");

state2 = 0;

state3 = 0;

count2 = 1;

alarm(WAITTIME);

data\_alarm = 0;

}

}

alarm(0);

printf("Response received.\n");

count2 = 1;

if (s == 0)

s = 1;

else

s = 0;

return length;

}

/\*\*

\* @brief Le informacao. Caso a informacao recebida esteja correta manda um mensagem de retorno de ready, caso contrario manda uma mensagem rej.

\*

\* @param fd Descritor do ficheiro

\* @param guardar2 Onde eu vou guardar a informacao. Ja deve vir previamente alocada memoria suficiente, do tamanho de Packat size.

\* @return int Retorna o comprimento da informacao lida

\*/

int llRead(int fd, unsigned char \*guardar2)

{

unsigned int bcc = 0, i = 0;

unsigned char receive;

unsigned char guardar[10 \* PackageSize];

state2 = 0;

int res;

int passei = 0;

int k = 0;

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

while (state2 != 6)

{

bcc = 0;

i = 0;

k = 0;

res = read(fd, &receive, 1);

if (res == 0)

continue;

changestate2Read(receive, 0);

passei++;

if (state2 != 0)

{

//Caso eu esteja a ler o primeiro byte de FLAG varias vezes, vou rejeitar este por ser o FLAG final da trama

if (passei != 1)

{

passei = 0;

continue;

}

passei = 0;

receive = 0;

// Receber todos os bytes da trama

while (receive != FLAG && i <= MaximumRead)

{

res = read(fd, &receive, 1);

if (res == 0)

continue;

guardar[i] = receive;

i++;

}

// Caso a trama tenho um tamanho muito pequeno, vai ser descartada

if (i < 4)

{

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

state2 = 0;

printf("Sending REJ message to Sender, Flag error...\n");

if (s == 0)

sendMessage("REJ0", "R", fd);

else

sendMessage("REJ1", "R", fd);

printf("REJ message sent. S: %d\n", s);

continue;

}

// Caso a trama tenha um tamanho demasiado grande

if (i > MaximumRead)

continue;

i = i - 3;

bcc = 0;

bcc ^= guardar[0];

bcc ^= guardar[1];

int j = 3, help = 0;

//Verificar se o bcc não esta stuffed

if (unStuff(guardar[2], guardar[3]) != 0)

j++;

if (unStuff(guardar[i], guardar[i + 1]) != 0)

{

i--;

help++;

}

changestate2Read(guardar[0], bcc);

changestate2Read(guardar[1], bcc);

if(state2!=3)continue;

if (unStuff(guardar[2], guardar[3]) == 0)

changestate2Read(guardar[2], bcc);

else

changestate2Read(unStuff(guardar[2], guardar[3]), bcc);

bcc = 0;

//Guardar a informacao na variavel de parametro

for (; j <= i; j++, k++)

{

unsigned int un = unStuff(guardar[j], guardar[j + 1]);

if (j != i)

{

if (un == 0)

{

bcc ^= guardar[j];

guardar2[k] = guardar[j];

}

else

{

bcc ^= un;

guardar2[k] = un;

j++;

}

}

else

{

bcc ^= guardar[j];

guardar2[k] = guardar[j];

}

}

// Verificar se a trama recebida não é repetida

if(comparer(guardar2,k)==0)

{

printf("Repeated message. Sending RR...\n");

if (s == 0)

sendMessage("RR0", "R", fd);

else

sendMessage("RR1", "R", fd);

printf("RR message sent. \n");

continue;

}

// Unstuff aos bytes finais

if (unStuff(guardar[i + help], guardar[i + 1 + help]) == 0)

changestate2Read(guardar[i + 1 + help], bcc);

else

changestate2Read(unStuff(guardar[i + help], guardar[i + 1 + help]), bcc);

changestate2Read(guardar[i + 2 + help], bcc);

// Caso a mensagem recebida esteja errada

if (state2 != 6)

{

printf("Sending REJ message to Sender. Message wrong...\n");

if (s == 0)

sendMessage("REJ0", "R", fd);

else

sendMessage("REJ1", "R", fd);

printf("REJ message sent. S:%d\n", s);

state2 = 0;

}

}

}

if (s)

s = 0;

else

s = 1;

// Caso a mensagem recebida esteja correta

printf("Data read. Sending RR message to Sender...\n");

if (s == 0)

sendMessage("RR0", "R", fd);

else

sendMessage("RR1", "R", fd);

printf("RR message sent. \n");

return k;

}

/\*\*

\* @brief Maquina de estados para o recetor. Caso estado 6, a mensagem recebida esta correta

\*

\* @param message Byte que eu recebi

\* @param bcc Bcc caso esteja no estado 3 ou 4

\*/

void changestate2Read(unsigned char message, unsigned char bcc)

{

switch (state2)

{

case 0:

if (message == FLAG)

state2 = 1;

break;

case 1:

if (message == Aemi)

state2 = 2;

else if (message != FLAG)

state2 = 0;

break;

case 2:

if (message == S0 && s == 0)

{

state2 = 3;

}

else if (message == S1 && s == 1)

{

state2 = 3;

}

else if (message == FLAG)

state2 = 1;

else

state2 = 0;

break;

case 3:

if (message == (bcc))

state2 = 4;

else if (message == FLAG)

state2 = 1;

else

state2 = 0;

break;

case 4:

if (message == (bcc))

state2 = 5;

else if (message == FLAG)

state2 = 1;

else

state2 = 0;

break;

case 5:

if (message == FLAG)

state2 = 6;

else

state2 = 0;

break;

default:

state2 = 0;

}

}

/\*\*

\* @brief Maquina de estados para a mensagem RR. Se no estado 5, a mensagem RR foi recebida com sucesso

\*

\* @param message Byte que eu recebi

\*/

void changestate2Write(unsigned char message)

{

switch (state2)

{

case 0:

if (message == FLAG)

state2 = 1;

break;

case 1:

if (message == Arec)

state2 = 2;

else if (message != FLAG)

state2 = 0;

break;

case 2:

if (message == Crr0 && s == 1)

{

state2 = 3;

}

else if (message == Crr1 && s == 0)

{

state2 = 3;

}

else if (message == FLAG)

state2 = 1;

else

state2 = 0;

break;

case 3:

if ((message == (Arec ^ Crr0) && s == 1) || (message == (Arec ^ Crr1) && s == 0))

state2 = 4;

else if (message == FLAG)

state2 = 1;

else

state2 = 0;

break;

case 4:

if (message == FLAG)

state2 = 5;

else

state2 = 0;

break;

default:

state2 = 0;

}

}

/\*\*

\* @brief Maquina de estados para a mensagem REJ. Caso estado 5 a mensagem REJ foi recebida com sucesso

\*

\* @param message Byte que eu recebi

\*/

void changestate2WriteREJ(unsigned char message)

{

switch (state3)

{

case 0:

if (message == FLAG)

state3 = 1;

break;

case 1:

if (message == Arec)

state3 = 2;

else if (message != FLAG)

state3 = 0;

break;

case 2:

if (message == Crej0 && s == 0)

{

state3 = 3;

}

else if (message == Crej1 && s == 1)

{

state3 = 3;

}

else if (message == FLAG)

state3 = 1;

else

state3 = 0;

break;

case 3:

if ((message == (Arec ^ Crej0) && s == 0) || (message == (Arec ^ Crej1) && s == 1))

state3 = 4;

else if (message == FLAG)

state3 = 1;

else

state3 = 0;

break;

case 4:

if (message == FLAG)

state3 = 5;

else

state3 = 0;

break;

default:

state3 = 0;

}

}

### Disconnect.c

#include "disconnect.h"

#include "shared.h"

#include "stateMachine.h"

#include "connect.h"

int count\_disconnect=0;

int disconnect\_alarm=0;

/\*\*

\* @brief Funcao chamada quando e recebido o sinal do alarm. Esta funcao só é chamada quando o programa esta a fazer disconnect

\*

\*/

void touchDisconnect()

{

disconnect\_alarm=1;

printf("Timeout number %d in DISC message response. Resending message...\n", count\_disconnect);

count\_disconnect++;

if(count\_disconnect>=4)

{

printf("Couldn't send the disconnect message. Exiting...\n");

exit(1);

}

}

/\*\*

\* @brief Faz a disconeção por parte do emissor

\*

\* @param fd Descritor do ficheiro

\*/

void disconnectSender(int fd)

{

int res;

signal(SIGALRM, touchDisconnect);

disconnect\_alarm=0;

count\_disconnect=0;

//Enviar a mensagem de disconnect para o Receiver

sendMessage("DISC","W",fd);

printf("Message DISC sent to Receiver.Waiting response...\n");

//Receber a mensagem de disconnect do Receiver

alarm(WAITTIME);

while(getStateDisc()!=5){

signal(SIGALRM, touchDisconnect);

unsigned char receive;

res=read(fd,&receive,1);

if(disconnect\_alarm)

{

disconnect\_alarm=0;

resetStates();

sendMessage("DISC","W",fd);

alarm(WAITTIME);

continue;

}

if(res==0)continue;

stateMachineDisc(receive);

}

alarm(0);

disconnect\_alarm=0;

printf("Received the DISC message from Receiver. Sending UA message.\n");

//Send UA message to Receiver

sendMessage("UA","W",fd);

printf("UA message sent\n");

}

/\*\*

\* @brief Faz a disconeção por parte do recetor

\*

\* @param fd Descritor do ficheiro

\*/

void disconnectReceiver(int fd)

{

unsigned char receive;

int res;

disconnect\_alarm=0;

count\_disconnect=0;

printf("Waiting for DISC message from Sender...\n");

//Receber a mensagem de disconnect do Emissor

while(getStateDisc()!=5){

res=read(fd,&receive,1);

if(res==0)continue;

stateMachineDisc2(receive);

}

//Send message to Emissor

printf("Sending DISC message to Sender...\n");

sendMessage("DISC","R",fd);

printf("DISC message sent to Sender. Waiting response...\n");

resetStates();

signal(SIGALRM, touchDisconnect);

alarm(WAITTIME);

while(getStateDisc()!=5){

signal(SIGALRM, touchDisconnect);

res=read(fd,&receive,1);

if(disconnect\_alarm)

{

disconnect\_alarm=0;

resetStates();

sendMessage("DISC","R",fd);

alarm(WAITTIME);

continue;

}

if(res==0)continue;

stateMachineUaDisc(receive);

}

alarm(0);

disconnect\_alarm=0;

printf("UA message received form Sender.\n");

}

/\*\*

\* @brief Fecha a ligação. Deve ser chamada a função setWrite ou setRead para indicar quem é o emissor ou o recetor

\*

\* @param fd Descrito do ficheiro de canal

\* @return int Retorna o 0 caso o canal tenha sido fechado com sucesso ou -1 caso contrario

\*/

int llClose(int fd)

{

if(getType())

disconnectReceiver(fd);

else

disconnectSender(fd);

sleep(2);

closeL(fd);

return close(fd);

}

### Reader.c

#include "includes.h"

#include "shared.h"

#include "stateMachine.h"

#include "data.h"

#include "connect.h"

#include "disconnect.h"

#include "testApplication.h"

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

{

if ( (argc != 2) || ((strcmp("/dev/ttyS0", argv[1])!=0) && (strcmp("/dev/ttyS1", argv[1])!=0) ))

{

printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS0\n");

exit(1);

}

setRead();

int fd=llOpen(argv[1]);

receiveData(fd);

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

llClose(fd);

return 0;

}

### Shared.c

#include "shared.h"

#include "connect.h"

struct termios \*oldtio,\*newtio;

int myType;

void setRead()

{

myType=1;

}

void setWrite()

{

myType=0;

}

int getType()

{

return myType;

}

/\*\*

\* Escreve um byte

\*/

int writeByte(int fd,unsigned char byte)

{

int res=write(fd,&byte,1);

return res;

}

void sendMessage(char \*type,char \* typeSender,int fd)

{

unsigned char send,a,c;

if(strcmp("W",typeSender)==0) a=Aemi;

else a=Arec;

if(strcmp("SET",type)==0) c=Cset;

else if(strcmp("UA",type)==0) c=Cua;

else if(strcmp("DISC",type)==0) c=Cdisc;

else if(strcmp("RR1",type)==0) c=Crr1;

else if(strcmp("RR0",type)==0) c=Crr0;

else if(strcmp("REJ1",type)==0) c=Crej1;

else if(strcmp("REJ0",type)==0) c=Crej0;

send=FLAG;

writeByte(fd,send);

send=a;

writeByte(fd,send);

send=c;

writeByte(fd,send);

send=a^c;

writeByte(fd,send);

send=FLAG;

writeByte(fd,send);

}

### StateMachine.c

#include "shared.h"

#include "stateMachine.h"

int state\_set=0;

int state\_ua=0;

int data\_state=0;

int disconnect\_state=0;

int getStateSet()

{

return state\_set;

}

int getStateUa()

{

return state\_ua;

}

int getStateDisc()

{

return disconnect\_state;

}

void resetStates()

{

state\_set=0;

state\_ua=0;

data\_state=0;

disconnect\_state=0;

}

//

void stateMachineSET(unsigned char message)

{

switch(state\_set){

case 0:

if(message == FLAG)

state\_set = 1;

break;

case 1:

if(message == Aemi)

state\_set = 2;

else if(message != FLAG)

state\_set = 0;

break;

case 2:

if(message == Cset)

state\_set = 3;

else if (message == FLAG)

state\_set = 1;

else

state\_set = 0;

break;

case 3:

if(message == (Aemi^Cset))

state\_set = 4;

else if (message == FLAG)

state\_set = 1;

else

state\_set = 0;

break;

case 4:

if(message == FLAG)

state\_set = 5;

else state\_set = 0;

break;

default: state\_set = 0;

}

}

//

void stateMachineUA(unsigned char message)

{

switch(state\_ua){

case 0:

if(message == FLAG)

state\_ua = 1;

break;

case 1:

if(message == Arec)

state\_ua = 2;

else if(message != FLAG)

state\_ua = 0;

break;

case 2:

if(message == Cua)

state\_ua = 3;

else if (message == FLAG)

state\_ua = 1;

else

state\_ua = 0;

break;

case 3:

if(message == (Arec^Cua))

state\_ua = 4;

else if (message == FLAG)

state\_ua = 1;

else

state\_ua = 0;

break;

case 4:

if(message == FLAG)

state\_ua = 5;

else state\_ua = 0;

break;

default: state\_ua = 0;

}

}

//

void stateMachineDisc(unsigned char message)

{

switch(disconnect\_state){

case 0:

if(message == FLAG)

disconnect\_state = 1;

break;

case 1:

if(message == Arec)

disconnect\_state = 2;

else if(message != FLAG)

disconnect\_state = 0;

break;

case 2:

if(message == Cdisc)

disconnect\_state = 3;

else if (message == FLAG)

disconnect\_state = 1;

else

disconnect\_state = 0;

break;

case 3:

if(message == (Arec^Cdisc))

disconnect\_state = 4;

else if (message == FLAG)

disconnect\_state = 1;

else

disconnect\_state = 0;

break;

case 4:

if(message == FLAG)

disconnect\_state = 5;

else disconnect\_state = 0;

break;

default: disconnect\_state = 0;

}

}

void stateMachineDisc2(unsigned char message)

{

switch(disconnect\_state){

case 0:

if(message == FLAG)

disconnect\_state = 1;

break;

case 1:

if(message == Aemi)

disconnect\_state = 2;

else if(message != FLAG)

disconnect\_state = 0;

break;

case 2:

if(message == Cdisc)

disconnect\_state = 3;

else if (message == FLAG)

disconnect\_state = 1;

else

disconnect\_state = 0;

break;

case 3:

if(message == (Aemi^Cdisc))

disconnect\_state = 4;

else if (message == FLAG)

disconnect\_state = 1;

else

disconnect\_state = 0;

break;

case 4:

if(message == FLAG)

disconnect\_state = 5;

else disconnect\_state = 0;

break;

default: disconnect\_state = 0;

}

}

void stateMachineUaDisc(unsigned char message)

{

switch(disconnect\_state){

case 0:

if(message == FLAG)

disconnect\_state = 1;

break;

case 1:

if(message == Aemi)

disconnect\_state = 2;

else if(message != FLAG)

disconnect\_state = 0;

break;

case 2:

if(message == Cua)

disconnect\_state = 3;

else if (message == FLAG)

disconnect\_state = 1;

else

disconnect\_state = 0;

break;

case 3:

if(message == (Aemi^Cua))

disconnect\_state = 4;

else if (message == FLAG)

disconnect\_state = 1;

else

disconnect\_state = 0;

break;

case 4:

if(message == FLAG)

disconnect\_state = 5;

else disconnect\_state = 0;

break;

default: disconnect\_state = 0;

}

}

### TestApplication.c

#include "testApplication.h"

#include "data.h"

int app\_stage=2;

int file\_size;

char my\_filename[255];

void sendData(unsigned char \*text, int fd,int seq,int leng){

unsigned char \* mandar=malloc(1+1+1+1+PackageSize);

int i=0;

mandar[i]=DataC;

i++;

mandar[i]=seq;

i++;

unsigned char L2=leng/256;

unsigned char L1=leng%256;

mandar[i]=L2;

i++;

mandar[i]=L1;

i++;

for(int j=0;i<leng+4;i++,j++)

{

mandar[i]=text[j];

}

llwrite(fd,mandar, PackageSize+4);

free(mandar);

return;

}

void applicationSend(int fd, char\* path){

FILE \* file;

file = fopen(path, "rb");

if(file == NULL)

{

perror("\n ERROR: File didn't open. \n");

return;

}

int size;

fseek(file,0,2);

size = ftell(file);

rewind(file);

char \*filename = basename(path);

sendControl(fd, size, filename, StartC); //start

printf("Control Tram start sent.\n");

unsigned char \*fileText = calloc(size,sizeof(unsigned char));

int ch;

unsigned long long int i=0;

while((ch = fgetc(file)) != EOF)

{

fileText[i]=ch;

i++;

}

unsigned char \*text=calloc(PackageSize+1,sizeof(unsigned char));

int j=1;

for(i=0;i<(sizeof(char)\*size)-PackageSize;i+=sizeof(char)\*PackageSize,j++)

{

for(int k=0;k<sizeof(char)\*PackageSize;k++)

{

text[k]=(\*fileText);

fileText+=sizeof(char);

}

sendData(text,fd,j,sizeof(char)\*PackageSize);

int percent=i\*100/size;

clear();

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

for(int progresso=1;progresso<=100;progresso++)

{

if(progresso<=percent)

{

printf("#");

}

else

printf(" ");

}

printf("|\n");

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

printf("Progress: %d%%\n",percent);

printf("Progress: %llu/%d\n",i,size);

}

i+=PackageSize;

for(int k=0;k<size-(i-sizeof(char)\*PackageSize);k++)

{

text[k]=(\*fileText);

fileText+=sizeof(char);

}

clear();

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

for(int progresso=1;progresso<=100;progresso++)

{

printf("#");

}

printf("|\n");

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

printf("Progress: 100%%\n");

printf("Progress: %d/%d\n",size,size);

sendData(text,fd,j,size-(i-sizeof(char)\*PackageSize));

printf("All data sent.\n");

sendControl(fd, size, filename, EndC);

printf("Control Tram end sent.\n");

fclose(file);

return;

}

void sendControl(int fd, int size, char\* filename, int startOrEnd){

char stringSize[255];

unsigned int typeSize = 0;

sprintf(stringSize, "%d", size);

unsigned int lengthSize = strlen(stringSize);

unsigned int typeName = 1;

unsigned int lengthName = strlen(filename);

//Tamanho do pacote de controlo

int pacoteSize = 1 + 1 + 1 + lengthSize + 1 + 1 +lengthName; //C + TLV1 + TLV2

unsigned char \*ctrl\_Tram = malloc(sizeof(unsigned char)\*pacoteSize);

ctrl\_Tram[0] = startOrEnd;

int i=1;

ctrl\_Tram[i] = typeSize;

i++;

ctrl\_Tram[i] = lengthSize;

i++;

for(int k=0;k<lengthSize;k++,i++)

{

ctrl\_Tram[i]=stringSize[k];

}

ctrl\_Tram[i] = typeName;

i++;

ctrl\_Tram[i] = lengthName;

i++;

for(int k=0;k<lengthName;k++,i++)

{

ctrl\_Tram[i]=filename[k];

}

printf("Sending Control.\n");

llwrite(fd,ctrl\_Tram, pacoteSize);

return;

}

void receiveDataRead(int fd)

{

unsigned long long int total=0;

unsigned char \*text= malloc(sizeof(unsigned char)\*(file\_size+1));

unsigned char \*lido=malloc(sizeof(unsigned char)\*(PackageSize+10));

while(total<file\_size)

{

llRead(fd,lido);

int i=2;

int L2=lido[i];

i++;

int L1=lido[i];

i++;

int numeroDeOctetos=L2\*256+L1;

for(int j=i;j<numeroDeOctetos+4 && total<file\_size;j++,total++)

{

text[total]=lido[j];

}

int percent=total\*100/file\_size;

clear();

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

for(int progresso=1;progresso<=100;progresso++)

{

if(progresso<=percent)

{

printf("#");

}

else

printf(" ");

}

printf("|\n");

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

printf("Progress: %d%%\n",percent);

printf("Progress: %llu/%d\n",total,file\_size);

}

int file=open(my\_filename,O\_CREAT | O\_WRONLY | O\_TRUNC,0600);

for(unsigned long long int i=0;i<file\_size;i++)

{

write(file,&(text[i]),1);

}

close(file);

}

void receiveControl(int fd, int startOrEnd)

{

unsigned char \*message=calloc(255,sizeof(unsigned char));

char takeOff[255];

llRead(fd,message);

int i=0;

if(message[i]==startOrEnd)

{

app\_stage=startOrEnd;

}

i++;

int sizeOcteto=0;

do

{

if(message[i]==0)

{

i++;

sizeOcteto=message[i];

i++;

int j=0;

for(;sizeOcteto>0;i++,sizeOcteto--,j++)

{

takeOff[j]=message[i];

}

takeOff[j]='\0';

sscanf(takeOff,"%d",&file\_size);

}

else if(message[i]==1)

{

i++;

sizeOcteto=message[i];

i++;

int j=0;

for(;sizeOcteto>0;i++,sizeOcteto--,j++)

{

takeOff[j]=message[i];

}

takeOff[j]='\0';

sscanf(takeOff,"%s",my\_filename);

}

}while(message[i]!=0);

printf("Filename: %s Size: %d \n",my\_filename,file\_size);

}

void receiveData(int fd){

receiveControl(fd,StartC);

printf("Received control tram start.\n");

receiveDataRead(fd);

printf("All data received.\n");

receiveControl(fd,EndC);

printf("Received control tram end.\n");

return;

}

### Writer.c

#include "includes.h"

#include "shared.h"

#include "stateMachine.h"

#include "data.h"

#include "connect.h"

#include "disconnect.h"

#include "testApplication.h"

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

{

if ( (argc != 3) || ((strcmp("/dev/ttyS0", argv[1])!=0) && (strcmp("/dev/ttyS1", argv[1])!=0) ))

{

printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS0\n");

exit(1);

}

setWrite();

int fd=llOpen(argv[1]);

applicationSend(fd,argv[2]);

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

llClose(fd);

return 0;

}