

Redes de Computadores

*1º Trabalho Laboratorial*

*Mestrado Integrado em Engenharia Informática e Computação*

*(30 de outubro de 2018)*

Bruno Sousa  **up201604145**@fe.up.pt

João Gonçalves **up201604245**@fe.up.pt

Pedro Neto **up201604420**@fe.up.pt

*Sumário*

Este relatório serve de complemento ao primeiro trabalho de Redes de Computadores. O trabalho consiste no desenvolvimento de um programa capaz de transmitir e receber ficheiros de um computador para o outro através de uma porta de série.

O trabalho foi concluído na totalidade, funcionando sem problemas a transmissão e receção de ficheiros sem perda de dados.

*Introdução*

O objetivo deste relatório é explicar a aplicação de envio de ficheiros de um computador para outro, através de uma porta de série. O relatório segue esta estrutura:

* **Arquitetura**

Apresentação dos blocos e interfaces da aplicação.

* **Estrutura do código**

Principais funções e sua relação com a arquitetura, macros e variáveis globais.

* **Casos de uso principais**

Identificação dos mesmos e demonstração das sequências de chamada de funções.

* **Protocolo de ligação lógica**

Identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código.

* **Protocolo de aplicação**

Identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código.

* **Validação**

Descrição dos testes efetuados com apresentação quantificada dos resultados.

* **Eficiência do protocolo de ligação de dados**

Caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.

* **Conclusão**

Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

*Arquitetura*

Sendo o trabalho o desenvolvimento de uma aplicação capaz de transmitir e receber dados, este pode ser dividido fundamentalmente em duas partes: funções de transmissão e funções de receção. Existe também uma divisão do código em dois ficheiros que separa as funções da camada de ligação de dados das funções da camada de aplicação.

Para correr a aplicação deve ser corrido o executável do lado do transmissor com os argumentos: “transmit”, a porta de série e o nome do ficheiro a ser enviado. Do lado do recetor, deve ser corrido com os argumentos: “receive” e a porta de série.

Para a execução “normal” da aplicação, esta deverá ser compilada com o comando make. Para a execução de testes de eficiência deverá ser compilada com o comando “make TEST]” em que TEST deverá ser o teste a ser efectuado. Testes possíveis são: efi\_size, efi\_baudrate, efi\_delay e efi\_error.

No modo “normal” será imprimida na consola uma barra de progresso e serão escritos no ficheiros “appLog.txt” e “llLog.txt” *logs* relativamente à parte da aplicação e à parte de ligação, respetivamente. Nos modos de teste, serão imprimidos apenas os resultados no ficheiro “testLog.txt”.

*Estrutura de código*

O código está dividido por dois ficheiros, existe o ficheiro *ll.c* (responsável pela camada de ligação) e o ficheiro *application.c* (responsável pela camada da aplicação). Existem ainda três ficheiros *header* contendo constantes do programa.

**Funções principais da camada de ligação (*ll.c*):**

* **setup –** faz todo o setup necessário para a conexão, abre a porta de série e retorna o *file desctiptor* desta,
* **llopen** – chama a função setup, e dependendo de uma *flag,* chama a função **llopenTransmitter** ou a função **llopenReceiver.** A função **llopenTransmitter** envia uma trama *SET* e recebe uma trama *UA*. A função **llopenReceiver** lê uma trama *SET* e envia uma trama *UA*.
* **llwrite** – recebe um *buffer* com dados da aplicação a enviar, transforma-os numa trama *I* realizando *stuffing*, envia-a utilizando a porta de série e espera por uma resposta. Caso a resposta seja *RR* apropriado, retorna. Caso contrário tenta reenviar a trama até receber uma resposta positiva.
* **llread** – lê da porta de série uma trama I e transforma-a de volta em dados da aplicação, realizando *destuffing*. Caso esteja tudo correto com o cabeçalho e com o *BCC2*, escreve na porta de serie uma trama *RR* apropriada e retorna à aplicação os dados. Caso contrário envia uma resposta *REJ* ou *RR* de outra trama e espera até receber uma trama I com tudo correto antes de devolver à aplicação os dados.
* **llclose** – dependo da flag que lhe é passada, chama a função **llcloseTransmitter** ou a função **llcloseReceiver**. A função **llcloseTransmitter** envia uma trama *DISC,* lê uma trama *DISC* e envia uma trama *UA*. A função **llcloseReceiver** lê uma trama *DISC* e envia uma trama *DISC*.

**Funções principais da camada da aplicação (*aplication.c*):**

* **main** – distribui o programa em transmissor ou recetor consoante o argumento passado, chamando a função **transmit** ou **receive**, respetivamente.
* **transmit** – função principal da transmissão de dados, chama **llopen**, envia através da função **sendControl** o pacote de controlo de abertura, lê e envia por **llwrite** o ficheiro dividindo-o em pacotes e preparando-os por **setDataPackage** para ser enviados como um pacote de dados, envia o através da função **sendControl** pacote de controlo de fecho, e chama **llclose**.
* **receive –** função principal da receção de dados, chama **llopen,** lê por **llread** os pacotes do ficheiro enviados pelo emissor, interpreta os pacotes pela função **interpretPacket**, escrevendo-os no novo ficheiro e chama **llclose**.

**Macros pertinentes:**

* **MAX\_ALARMS** – número de alarmes até sair da aplicação caso não receba uma resposta.
* **TIMEOUT** – número de segundos de cada alarme.
* **PACKET\_SIZE** – tamanho de cada pacote de dados.
* **BAUDRATE –** capacidade da ligação.

**Variáveis globais:**

* **trama** – número sequencial da trama (*Ns*) a enviar, inicializada a 0. Vai variando entre 0 e 1 de acordo com a trama que irá enviar (do lado de quem envia) e de acordo com a trama que espera receber (do lado do recetor).
* **flagAlarm** – *TRUE* quando o alarme está ativo, *FALSE* caso esteja inativo.
* **conta\_alarme** – contador de alarmes, inicializada a 0. É incrementado sempre que é necessário reenviar algo por falta de resposta (volta a 0 quando/caso chegue a resposta). Se chegar a MAX\_ALARMS, a aplicação fecha.

*Casos de Uso*

A aplicação permite, através de dois modos de utilização enviar e receber um ficheiro (ex. pinguim.gif) de um computador para o outro por uma porta de série.

A transmissão de dados dá-se com a seguinte sequência:

* Chamada de **llopen** para abrir a porta de série.
* Abertura do ficheiro pretendido.
* Envio do pacote de controlo *START* com informação sobre o nome e tamanho do ficheiro, através da função **sendControl**.
* Ciclo em que são lidos dados do ficheiro, é construído um pacote de dados através da função **setDataPackage**, e é enviado para a porta de série através da função **llwrite**.
* Quando o ficheiro está completamente lido, é enviado um pacote de controlo *END* através da função **sendControl**.
* A aplicação acaba chamando **llclose** para fechar a porta de série.

A receção de dados dá-se com a seguinte sequência:

* Chamada de **llopen** para abrir a porta de série.
* Leitura do pacote de controlo *START* através de **llread**, que é interpretado através da função **interpretPacket**, criando o ficheiro com o nome recebido.
* Ciclo de leitura de pacotes de dados através de **llread**, interpretados por **interpretPacket**, escrevendo os dados recebidos no ficheiro.
* Eventualmente, será recebido o pacote de controlo *END*, terminando o ciclo de leitura.
* A aplicação acaba chamando **llclose** para fechar a porta de série.

*Protocolo de ligação lógica*

**LLOPEN**

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

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

Em ambos, chama a função **setup** que abre a porta de série cujo nome é port, e retorna um *file descriptor*.

No emissor, esta função envia a trama de controlo *SET* e ativa o temporizador que é desativado depois de receber resposta (*UA*). Se não receber resposta dentro de um tempo *time-out*, *SET* é reenviado. Este mecanismo de retransmissão só é repetido um número máximo de vezes, se este número for atingido o programa termina.

No recetor, esta função espera pela chegada de uma trama de controlo SET para responder com um UA.

A leitura é feita dentro de um ciclo que só termina caso seja lida a trama pretendida ou que esgote o seu tempo definido pelos alarmes.

As escritas são feitas trama a trama, no entanto a leitura é feita carater a carater.

**LLWRITE**

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

Esta é a função no emissor responsável pelo envio das tramas e pelo *stuffing* das mesmas.

Primeiro são organizados os dados em tramas (*framing*), ou seja, acrescentado o cabeçalho do Protocolo de Ligação à mensagem. Depois é feito o *stuffing* da mensagem e do *BCC2*. A trama fica pronta para ser enviada trama a trama.

O envio da trama tem o mesmo mecanismo de *time-out* e retransmissão que o envio do *SET* no ***llopen***. Ou seja, depois de enviar a trama é acionado um alarme até à receção de uma resposta (*RR* ou *REJ*) e se atingido esse alarme a mensagem é reenvida (mecanismo que se pode ocorrer um número máximo de vezes). Se recebido um *REJ* ou um *RR* da mesma trama a mensagem é reenviada.

**LLREAD**

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

Esta é a função no recetor responsável pela receção das tramas.

Na função **checkInitials**, verifica se o cabeçalho da trama está correto.

É depois exectuado o *destuffing*, verificado o *BCC2*, caso esteja correto é enviado *RR*, caso contrário *REJ*. A trama de resposta é enviada através da função **sendRR**. O campo de controlo enviado depende do número de sequência da trama.

**LLCLOSE**

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

Esta função tem a responsabilidade de terminar a ligação entre o emissor e o recetor. É dividida em duas funções: **llcloseTransmitter** e **llcloseReceiver**, consoante o valor de *flag*.

No emissor, é enviado a trama de Supervisão *DISC* e esperado outro *DISC* de resposta. Para finalizar é enviado um *UA*.

No recetor é esperado um *DISC*, enviado um *DISC* e esperado um *UA*.

Em ambas é fechada a porta de série.

*Protocolo de aplicação*

O protoloco de aplicação implementado tem como aspetos principais:

* O envio dos pacotes de controlo START e END. Estes contêm o nome e o tamanho do ficheiro a ser enviado;
* A divisão do ficheiro em pacotes quando se trata do emissor e a concatenação dos pacotes recebidos, quando se trata do recetor;
* Encapsular cada pacote de dados com um cabeçalho contendo o número de sequência do pacote (módulo 255) e o tamanho do pacote;
* Leitura do ficheiro a enviar, quando se trata do emissor, e criação do ficheiro, quando se trata do recetor.

Estas funcionalidades foram implementadas usando funções descritas a seguir.

**transmit:**

int **transmit**(char \*port, char \*file, int packet\_size); //Packet\_size //apenas para testes

Esta função chama **llopen** para abrir a porta de série e iniciar a comunicação, abre o ficheiro a ser transmitido, envia um pacote de controlo *START* usando a função **sendControl**, executa um ciclo em que lê data do ficheiro, chama a função **setDataPackage** para construir um pacote de dados a partir desses dados e envia usando a função **llwrite**. Quando o ficheiro foi totalmente enviado, é enviado um pacote de controlo *END* e chamda a função **llclose** para terminar a comunicação e fechar a porta de série.

**receive:**

int **receive**(char \*port, int packet\_size); //Packet\_size apenas para //testes

Esta função chama **llopen** para abrir a porta de série e iniciar a comunicação, depois lê o pacote de controlo *START* utilizando **llread**, interpreta-o em **interpretPacket**, onde cria o ficheiro a ser recebido. Posteriormente, executa um ciclo onde lê pacotes de dados de **llread**, interpreta-os com **interprePacket** e os escreve no ficheiro. Eventualmente, interpretará um pacote de controlo *END*, o que fará a leitura parar. Posteriormente, chamará a função **llclose** para terminar a comunicação e fechar a porta de série.

*Validação*

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

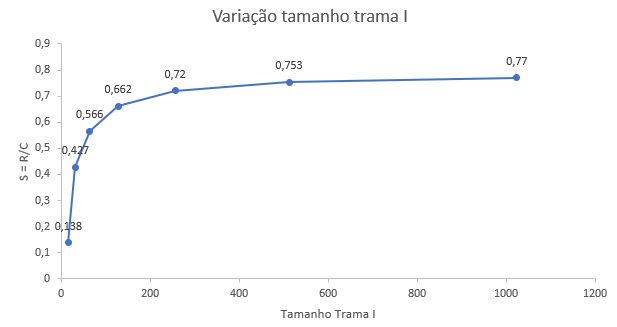
* Envio de ficheiros de vários tamanhos.
* Geração de curto circuito enquanto se envia um ficheiro.
* Interrupção da ligação por alguns segundos enquanto se envia um ficheiro.
* Envio do um ficheiro com variação do tamanho de pacotes.

Todos os testes foram concluídos com sucesso.

*Eficiência*

Nota: Código de teste de eficiência elaborado após a apresentação.

De forma a testar a eficiência foram realizados 4 testes:

* Variação do tamanho das tramas I

Variando o tamanho de pacotes de dados, foi possível variar o tamanho da trama I. Como podemos ver pelo gráfico, quanto maior for a trama I, mais eficiente será o protocolo da ligação de dados.

.

* Variação da capacidade de ligação

Alterando o *BAUDRATE*, foi possível alterar a capacidade de ligação.

Como podemos ver pelo gráfico, quanto maior a capacidade de ligação, menor será a eficiência.

* Geração de atraso

Introduzindo atrasos no processamento de cada trama recebida

diminuirá a eficiência.

* Geração aleatória de erros

Introduzindo erros tanto no cabeçalho como no campo de dados das

tramas I diminuirá a eficiência. Os erros no cabeçalho terão um maior impacto pois quando é detetado um erro no campo de dados é enviado um *REJ* que fará com que o transmissor reenvie imediatamente a trama, enquanto que quando é detetado um erro no cabeçalho é necessário esperar que o transmissor receba um alarme e reenvie a trama.

*Conclusões*

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

Foram alcançados os objetivos de aprendizagem propostos, fazendo o envio de dados assíncrono dividido por camadas.

*Anexos*

**application.c**

#include "common.h"

#include "ll.h"

#include <stdio.h>

#include <fcntl.h>

#include <stdlib.h>

#include <math.h>

#include <unistd.h>

#include <string.h>

void sendControl(int porta, char\* name, int size, int option) {

    int package\_size = 5;

    int name\_length = strlen(name) + 1;

    int byte\_size = ceil(log2((double)size+1.0)/8);

    package\_size += byte\_size;

    package\_size += name\_length;

    char \*package;

    package = (char \*)malloc(package\_size);

    package[0] = option;

    package[1] = 0x00;

    package[2] = byte\_size;

    int z=0;

    int i=byte\_size+2;

    for(; i>2;i--)

    {

        package[i]= (size >> (8\*(z))) & 0xFF;

        z++;

    }

    i=3+byte\_size;

    package[i]=0x01;

    package[i+1]=name\_length;

    int j;

    for(j=0;j<name\_length;j++)

    {

        package[i+j+2]= name[j];

    }

    llwrite(porta, package, package\_size);

    free(package);

}

void setDataPackage(char\* buf, int data\_size, int n) {

    char\* data = (char \*)malloc(data\_size);

   memcpy(data, buf, data\_size);

    buf[0] = C\_DATA;

    buf[1] = n % 256;

    buf[2] = data\_size / 256;

    buf[3] = data\_size % 256;

    int i;

    for (i = 0; i < data\_size; i++)

    {

        buf[i+4] = data[i];

    }

    free(data);

}

int transmit(char \*port, char \*file)

{

    int serial = llopen(port, TRANSMITTER);

if (serial < 0)

return -1;

    FILE\* ficheiro = fopen(file, "r");

    if (ficheiro == 0) {

        printf("Error: %s is not a file.\n", file);

        return -1;

}

    fseek(ficheiro, 0L, SEEK\_END);

    int size = ftell(ficheiro)-1;

    fclose(ficheiro);

    int n = 0; //PACKET n = 0 será START, a partir de n = 1 dados

    sendControl(serial, file, size, C\_START);

    n++;

    int fd = open(file,O\_RDONLY);

    int res;

    char buf[PACKET\_SIZE];

    while(res != 0)

    {

        res = read(fd, buf, PACKET\_SIZE - 4);

        if(res == 0)

            break;

        setDataPackage(buf, res, n);

        if (llwrite(serial, buf, res + 4) < 0)

            return -1;

        n++;

    }

    sendControl(serial, file, size, C\_END);

    close(fd);

    return llclose(serial, TRANSMITTER);

}

int interpretPacket(char\* buf, int res, int\* file, int n)

{

    if(buf[0]==C\_START)

    {

        int i=5 + buf[2];

        int j=0;

        char\* file\_name = (char \*)malloc(res-i);

        for(;i<res;i++, j++)

        {

            file\_name[j]=buf[i];

        }

        int fd = open(file\_name, O\_WRONLY | O\_CREAT | O\_TRUNC , 0644);

        \*file = fd;

        free(file\_name);

        return FALSE;

    }

    if(buf[0]==C\_END)

        return TRUE;

    if(buf[0]!=C\_DATA)

        printf("Campo Controlo Packet %d não conhecido, assumindo 1.\n", n);

    if(buf[0]==C\_DATA)

    {

        char\* data = (char \*)malloc(res - 4);

        int i;

        for (i = 0; i < res - 4; i++)

        {

            data[i] = buf[i+4];

        }

        write(\*file, data, res - 4);

        free(data);

        return FALSE;

    }

    return FALSE;

}

int receive(char \*port)

{

int serial = llopen(port, RECEIVER);

if (serial < 0)

return -1;

char buffer[PACKET\_SIZE];

int res = llread(serial, buffer);

int file;

    int n = 0;

    interpretPacket(buffer, res, &file, n); //PACKET n = 0 será START, a partir de n = 1 dados

    n++;

int end = FALSE;

while(!end)

{

res = llread(serial, buffer);

        if(res < 0)

            return res;

end = interpretPacket(buffer, res, &file, n);

        n++;

}

    close(file);

return llclose(serial, RECEIVER);

}

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

{

if (argc < 2 || argc > 4)

{

printf("Usage: [transmit/receive] SerialPort [filename]\n");

return -1;

}

if (argc == 4)

{

if (strcmp(argv[1],"transmit"))

{

printf("Usage: [transmit/receive] SerialPort [filename]\n");

return -1;

}

return transmit(argv[2], argv[3]);

}

    if (argc == 3)

{

if (strcmp(argv[1],"receive"))

{

printf("Usage: [transmit/receive] SerialPort [filename]\n");

return -1;

}

return receive(argv[2]);

}

    printf("Usage: [transmit/receive] SerialPort [filename]\n");

return 1;

}

**ll.c:**

#include "common.h"

#include "constants.h"

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <termios.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <strings.h>

#include <signal.h>

int trama = 0;

int flag\_alarme = 0;

int conta\_alarme = 0;

void atende\_alarme()

{

    flag\_alarme = 1;

    conta\_alarme++;

    printf("Alarme %d\n", conta\_alarme);

}

void desativa\_alarme()

{

    flag\_alarme = 0;

    alarm(0);

}

int setup(char\* port)

{

int fd;

struct termios oldtio, newtio;

if (((strcmp("/dev/ttyS0", port) != 0) && (strcmp("/dev/ttyS1", port) != 0)))

{

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

exit(1);

}

/\*

Open serial port device for reading and writing and not as controlling tty

because we don't want to get killed if linenoise sends CTRL-C.

\*/

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

if (fd < 0)

{

perror(port);

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] = 1; /\* inter-character timer unused \*/

newtio.c\_cc[VMIN] = 0; /\* blocking read until 5 chars received \*/

/\*

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

leitura do(s) próximo(s) caracter(es)

\*/

tcflush(fd, TCIOFLUSH);

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

{

perror("tcsetattr");

exit(-1);

}

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

    struct sigaction action;

action.sa\_handler = atende\_alarme;

sigemptyset(&action.sa\_mask);

action.sa\_flags = 0;

sigaction(SIGALRM, &action, NULL);

return fd;

}

int llopenTransmitter(int fd)

{

    char set[SET\_SIZE];

    set[0] = FLAG;

    set[1] = SET\_ADDRESS;

    set[2] = SET\_CONTROL;

    set[3] = SET\_ADDRESS ^ SET\_CONTROL;

    set[4] = FLAG;

    char ua[UA\_SIZE];

    int recebido = FALSE;

    int i = 0, res = 0;

    while (conta\_alarme <= MAX\_ALARMS && !recebido)

    {

        desativa\_alarme();

        res = write(fd, set, SET\_SIZE);

        if (res != SET\_SIZE)

            continue;

        printf("SET enviado!\n");

        alarm(TIMEOUT);

        i = 0;

        while (!flag\_alarme && !recebido)

        {

            res = read(fd, ua + i, 1);

            if (res <= 0)

                continue;

            switch (i)

            {

            case 0:

                if (ua[i] != FLAG)

                    continue;

                break;

            case 1:

                if (ua[i] != UA\_ADDRESS\_SENDER)

                {

                    if (ua[i] != FLAG)

                        i = 0;

                    continue;

                }

                break;

            case 2:

                if (ua[i] != UA\_CONTROL)

                {

                    if (ua[i] != FLAG)

                        i = 0;

                    else

                        i = 1;

                    continue;

                }

                break;

            case 3:

                if (ua[i] != (UA\_ADDRESS\_SENDER ^ UA\_CONTROL))

                {

                    if (ua[i] != FLAG)

                        i = 0;

                    else

                        i = 1;

                    continue;

                }

                break;

            case 4:

                if (ua[i] != FLAG)

                {

                    i = 0;

                    continue;

                }

                break;

            default:

                break;

            }

            i++;

            if (i == UA\_SIZE)

            {

                recebido = TRUE;

                printf("UA recebido!\n");

                desativa\_alarme();

            }

        }

    }

    if(!recebido)

        return -1;

    conta\_alarme = 0;

    return 0;

}

int llopenReceiver(int fd)

{

char set[SET\_SIZE];

char ua[UA\_SIZE];

ua[0] = FLAG;

ua[1] = UA\_ADDRESS\_SENDER;

ua[2] = UA\_CONTROL;

ua[3] = UA\_ADDRESS\_SENDER ^ UA\_CONTROL;

ua[4] = FLAG;

int recebido = FALSE;

int i = 0, res = 0;

while (!recebido)

{

res = read(fd, set + i, 1);

if (res <= 0)

continue;

switch (i)

{

case 0:

if (set[i] != FLAG)

continue;

break;

case 1:

if (set[i] != SET\_ADDRESS)

{

if (set[i] != FLAG)

i = 0;

continue;

}

break;

case 2:

if (set[i] != SET\_CONTROL)

{

if (set[i] != FLAG)

i = 0;

else

i = 1;

continue;

}

break;

case 3:

if (set[i] != (SET\_ADDRESS ^ SET\_CONTROL))

{

if (set[i] != FLAG)

i = 0;

else

i = 1;

continue;

}

break;

case 4:

if (set[i] != FLAG)

{

i = 0;

continue;

}

break;

default:

break;

}

i++;

if (i == SET\_SIZE)

{

recebido = TRUE;

printf("SET recebido!\n");

}

}

int enviado = FALSE;

while (!enviado)

{

res = write(fd, ua, UA\_SIZE);

printf("UA enviado!\n");

if (res == UA\_SIZE)

enviado = TRUE;

}

    return 0;

}

int llopen(char \*port, int flag)

{

    int fd = setup(port);

    int res;

    if (flag == TRANSMITTER)

        res = llopenTransmitter(fd);

    else if (flag == RECEIVER)

        res = llopenReceiver(fd);

    if (res < 0)

        return res;

    return fd;

}

int check\_initials(int fd)

{

char inf[4];

int recebido = FALSE;

int i = 0, res = 0;

    int temp\_trama = -1;

while (!recebido)

{

res = read(fd, inf + i, 1);

if (res <= 0)

continue;

switch (i)

{

case 0:

if (inf[i] != FLAG)

continue;

break;

case 1:

if (inf[i] != INF\_ADDRESS)

{

if (inf[i] != FLAG)

i = 0;

continue;

}

break;

case 2:

            if (inf[i] == INF\_CONTROL0)

                    temp\_trama = 0;

            else if (inf[i] == INF\_CONTROL1)

                    temp\_trama = 1;

else

{

if (inf[i] != FLAG)

i = 0;

else

i = 1;

continue;

}

break;

case 3:

if (!((inf[i] == (INF\_ADDRESS ^ INF\_CONTROL0) && temp\_trama == 0) || (inf[i] == (INF\_ADDRESS ^ INF\_CONTROL1) && temp\_trama == 1)))

{

if (inf[i] != FLAG)

i = 0;

else

i = 1;

continue;

}

break;

default:

break;

}

i++;

if (i == INF\_INIT\_SIZE)

recebido = TRUE;

}

    return temp\_trama;

}

void sendRR(int fd, int rej){

    char rr[5];

rr[0] = FLAG;

rr[1] = RR\_ADDRESS;

    if(trama == 0)

    {

        if(rej)

        {

            rr[2] = REJ\_CONTROL0;

        rr[3] = RR\_ADDRESS ^ REJ\_CONTROL0;

        }

        else

        {

            rr[2] = RR\_CONTROL0;

        rr[3] = RR\_ADDRESS ^ RR\_CONTROL0;

        }

    }

    else

    {

        if(rej)

        {

            rr[2] = REJ\_CONTROL1;

        rr[3] = RR\_ADDRESS ^ REJ\_CONTROL1;

        }

        else

        {

            rr[2] = RR\_CONTROL1;

        rr[3] = RR\_ADDRESS ^ RR\_CONTROL1;

        }

    }

rr[4] = FLAG;

    int enviado = FALSE;

    int res;

while (!enviado)

{

res = write(fd, rr, RR\_SIZE);

if (res == RR\_SIZE)

enviado = TRUE;

}

}

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

{

    if (length <= 0 || !(trama == 0 || trama == 1))

        return -1;

    char bcc2 = 0;

    int i = 0;

    for (; i < length; i++)

        bcc2 ^= buffer[i];

    char \*buf;

    buf = (char \*)malloc((length + 1) \* 2 + 5);

    buf[0] = FLAG;

    buf[1] = INF\_ADDRESS;

    if (trama == 0)

    {

        buf[2] = INF\_CONTROL0;

        buf[3] = INF\_ADDRESS ^ INF\_CONTROL0;

    }

    else if (trama == 1)

    {

        buf[2] = INF\_CONTROL1;

        buf[3] = INF\_ADDRESS ^ INF\_CONTROL1;

    }

    i = 0;

    int j = 4;

    for (; i < length; i++, j++)

    {

        if (buffer[i] == FLAG)

        {

            buf[j] = INF\_ESCAPE;

            j++;

            buf[j] = INF\_XOR\_FLAG;

        }

        else if (buffer[i] == INF\_ESCAPE)

        {

            buf[j] = INF\_ESCAPE;

            j++;

            buf[j] = INF\_XOR\_ESCAPE;

        }

        else

            buf[j] = buffer[i];

    }

    if (bcc2 == FLAG)

    {

        buf[j] = INF\_ESCAPE;

        j++;

        buf[j] = INF\_XOR\_FLAG;

    }

    else if (bcc2 == INF\_ESCAPE)

    {

        buf[j] = INF\_ESCAPE;

        j++;

        buf[j] = INF\_XOR\_ESCAPE;

    }

    else

        buf[j] = bcc2;

    j++;

    buf[j] = FLAG;

    j++; // j contém numero de chars usados

    char rr[5];

    int res;

    int recebido = FALSE;

    i = 0;

    int temp\_trama = -1;

    int rej = FALSE;

    while (conta\_alarme <= MAX\_ALARMS && !recebido)

    {

        desativa\_alarme();

        res = write(fd, buf, j);

        if (res != j)

            continue;

        printf("Trama I%d enviada!\n", trama);

        alarm(TIMEOUT);

        i = 0;

        while (!flag\_alarme && !recebido)

        {

            res = read(fd, rr + i, 1);

            if (res <= 0)

                continue;

            switch (i)

            {

            case 0:

                if (rr[i] != FLAG)

                    continue;

                break;

            case 1:

                if (rr[i] != RR\_ADDRESS)

                {

                    if (rr[i] != FLAG)

                        i = 0;

                    continue;

                }

                break;

            case 2:

                if (rr[i] == (char)RR\_CONTROL0)

                    temp\_trama = 0;

                else if (rr[i] == (char)RR\_CONTROL1)

                    temp\_trama = 1;

                else if (rr[i] == (char)REJ\_CONTROL0)

                {

                    rej = TRUE;

                    temp\_trama = 0;

                }

                else if (rr[i] == (char)REJ\_CONTROL1)

                {

                    rej = TRUE;

                    temp\_trama = 1;

                }

                else

                {

                    if (rr[i] != FLAG)

                        i = 0;

                    else

                        i = 1;

                    continue;

                }

                break;

            case 3:

                if ((rej && ((rr[i] == (char)(RR\_ADDRESS ^ REJ\_CONTROL0) && temp\_trama == 0) || (rr[i] == (char)(RR\_ADDRESS ^ REJ\_CONTROL1) && temp\_trama == 1))) || (!rej && ((rr[i] == (char)(RR\_ADDRESS ^ RR\_CONTROL0) && temp\_trama == 0) || (rr[i] == (char)(RR\_ADDRESS ^ RR\_CONTROL1) && temp\_trama == 1))))

                break;

                else

                {

                    if (rr[i] != FLAG)

                        i = 0;

                    else

                        i = 1;

                    continue;

                }

            case 4:

                if (rr[i] != FLAG)

                {

                    i = 0;

                    continue;

                }

                break;

            default:

                break;

            }

            i++;

            if (i == RR\_SIZE)

            {

                desativa\_alarme();

                recebido = TRUE;

                if(rej)

                    printf("REJ%d recebido!\n", temp\_trama);

                else printf("RR%d recebido!\n", temp\_trama);

            }

        }

        if(!recebido)

            continue;

        conta\_alarme = 0;

        if(rej || trama == temp\_trama)

        {

            temp\_trama = -1;

            recebido = FALSE;

            rej = FALSE;

            printf("Re-sending trama I%d!\n",trama);

            continue;

        }

        if (trama == 0)

            trama = 1;

        else trama = 0;

    }

    free(buf);

    return j;

}

int llread(int fd, char \*buffer)

{

    int certo = FALSE;

    int rej;

    int temp\_trama;

    char data;

    char bcc;

    int recebido;

    int i;

    int destuffing;

    int res;

    while(!certo)

    {

        temp\_trama = check\_initials(fd);

        if (temp\_trama < 0)

         return temp\_trama;

        rej = FALSE;

        recebido = FALSE;

        i = 0;

        destuffing = FALSE;

        res = 0;

        while(!recebido)

        {

         res = read(fd,&data,1);

         if (res <= 0)

         continue;

            if (destuffing)

            {

                destuffing = FALSE;

                if(data == INF\_XOR\_FLAG)

                    data = FLAG;

                else if(data == INF\_XOR\_ESCAPE)

                    data = INF\_ESCAPE;

                else return -1;

            }

         else if (data == FLAG)

         {

         if(i == 0)

         return -1;

         recebido = TRUE;

         break;

         }

         //DE-STUFFING

            else if (data == INF\_ESCAPE)

            {

                destuffing = TRUE;

                continue;

            }

         if (i != 0)

         buffer[i-1] = bcc;

         bcc = data;

         i++;

        }

        i--; //BCC doesn't count

        //i has num char read to buffer

        char check = 0;

        int j = 0;

        for(; j < i; j++)

            check ^= buffer[j];

        if (check != bcc)

         rej = TRUE;

        printf("Trama %d recebida!\n", temp\_trama);

        if(rej && temp\_trama == trama)

        {

            sendRR(fd,rej);

            printf("REJ%d enviado!\n", trama);

            continue;

        }

        if(temp\_trama != trama)

        {

            sendRR(fd,rej);

            printf("RR%d re-enviado!\n", trama);

            continue;

        }

        certo = TRUE;

        if(trama == 0)

            trama = 1;

        else trama = 0;

        sendRR(fd,rej);

        printf("RR%d enviado!\n", trama);

    }

return i;

}

int llcloseTransmitter(int fd)

{

    char disc\_sender[5];

    disc\_sender[0] = FLAG;

    disc\_sender[1] = DISC\_ADDRESS\_SENDER;

    disc\_sender[2] = DISC\_CONTROL;

    disc\_sender[3] = DISC\_ADDRESS\_SENDER ^ DISC\_CONTROL;

    disc\_sender[4] = FLAG;

    char disc\_receiver[5];

    int recebido = FALSE;

    int i = 0, res = 0;

    while (conta\_alarme <= MAX\_ALARMS && !recebido)

    {

        desativa\_alarme();

        res = write(fd, disc\_sender, DISC\_SIZE);

        if (res != DISC\_SIZE)

            continue;

        printf("DISC enviado!\n");

        alarm(TIMEOUT);

        i = 0;

        while (!flag\_alarme && !recebido)

        {

            res = read(fd, disc\_receiver + i, 1);

            if (res <= 0)

                continue;

            switch (i)

            {

            case 0:

                if (disc\_receiver[i] != FLAG)

                    continue;

                break;

            case 1:

                if (disc\_receiver[i] != DISC\_ADDRESS\_RECEIVER)

                {

                    if (disc\_receiver[i] != FLAG)

                        i = 0;

                    continue;

                }

                break;

            case 2:

                if (disc\_receiver[i] != DISC\_CONTROL)

                {

                    if (disc\_receiver[i] != FLAG)

                        i = 0;

                    else

                        i = 1;

                    continue;

                }

                break;

            case 3:

                if (disc\_receiver[i] != (DISC\_ADDRESS\_RECEIVER ^ DISC\_CONTROL))

                {

                    if (disc\_receiver[i] != FLAG)

                        i = 0;

                    else

                        i = 1;

                    continue;

                }

                break;

            case 4:

                if (disc\_receiver[i] != FLAG)

                {

                    i = 0;

                    continue;

                }

                break;

            default:

                break;

            }

            i++;

            if (i == DISC\_SIZE)

            {

                recebido = TRUE;

                printf("DISC recebido!\n");

                desativa\_alarme();

            }

        }

    }

    if(!recebido)

        return -1;

    conta\_alarme = 0;

    char ua[5];

ua[0] = FLAG;

ua[1] = UA\_ADDRESS\_RECEIVER;

ua[2] = UA\_CONTROL;

ua[3] = UA\_ADDRESS\_RECEIVER ^ UA\_CONTROL;

ua[4] = FLAG;

    int enviado = FALSE;

while (!enviado)

{

res = write(fd, ua, UA\_SIZE);

printf("UA enviado!\n");

if (res == UA\_SIZE)

enviado = TRUE;

}

    close(fd);

    return 1;

}

int llcloseReceiver(int fd)

{

char disc\_sender[5];

char disc\_receiver[5];

disc\_receiver[0] = FLAG;

disc\_receiver[1] = DISC\_ADDRESS\_RECEIVER;

disc\_receiver[2] = DISC\_CONTROL;

disc\_receiver[3] = DISC\_ADDRESS\_RECEIVER ^ DISC\_CONTROL;

disc\_receiver[4] = FLAG;

int recebido = FALSE;

int i = 0, res = 0;

while (!recebido)

{

res = read(fd, disc\_sender + i, 1);

if (res <= 0)

continue;

switch (i)

{

case 0:

if (disc\_sender[i] != FLAG)

continue;

break;

case 1:

if (disc\_sender[i] != DISC\_ADDRESS\_SENDER)

{

if (disc\_sender[i] != FLAG)

i = 0;

continue;

}

break;

case 2:

if (disc\_sender[i] != DISC\_CONTROL)

{

if (disc\_sender[i] != FLAG)

i = 0;

else

i = 1;

continue;

}

break;

case 3:

if (disc\_sender[i] != (DISC\_ADDRESS\_SENDER ^ DISC\_CONTROL))

{

if (disc\_sender[i] != FLAG)

i = 0;

else

i = 1;

continue;

}

break;

case 4:

if (disc\_sender[i] != FLAG)

{

i = 0;

continue;

}

break;

default:

break;

}

i++;

if (i == DISC\_SIZE)

{

recebido = TRUE;

printf("DISC recebido!\n");

}

}

int enviado = FALSE;

while (!enviado)

{

res = write(fd, disc\_receiver, DISC\_SIZE);

printf("DISC enviado!\n");

if (res == DISC\_SIZE)

enviado = TRUE;

}

    close(fd);

    return 1;

}

int llclose(int fd, int flag)

{

    if(flag == TRANSMITTER)

        return llcloseTransmitter(fd);

    if(flag == RECEIVER)

        return llcloseReceiver(fd);

    return -1;

}

**ll.h:**

#ifndef LL\_H

#define LL\_H

#define C\_START 0x02

#define C\_END 0x03

#define C\_DATA 0x01

#define PACKET\_SIZE 100

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

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

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

int llclose(int fd, int flag);

#endif

**constants.h:**

#ifndef CONSTANTS\_H

#define CONSTANTS\_H

#define BAUDRATE B38400

#define MODEMDEVICE "/dev/ttyS1"

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

#define FLAG 0x7E

#define SET\_SIZE 5

#define SET\_ADDRESS 0x03

#define SET\_CONTROL 0x03

#define UA\_SIZE 5

#define UA\_ADDRESS\_SENDER 0x03

#define UA\_ADDRESS\_RECEIVER 0x01

#define UA\_CONTROL 0x07

#define INF\_INIT\_SIZE 4

#define INF\_ADDRESS 0x03

#define INF\_CONTROL0 0x00

#define INF\_CONTROL1 0x40

#define INF\_ESCAPE 0x7D

#define INF\_XOR\_FLAG 0x5E

#define INF\_XOR\_ESCAPE 0x5D

#define RR\_SIZE 5

#define RR\_ADDRESS 0x03

#define RR\_CONTROL0 0x05

#define RR\_CONTROL1 0x85

#define REJ\_CONTROL0 0x01

#define REJ\_CONTROL1 0x81

#define DISC\_SIZE 5

#define DISC\_ADDRESS\_SENDER 0x03

#define DISC\_ADDRESS\_RECEIVER 0x01

#define DISC\_CONTROL 0x0B

#define MAX\_ALARMS 3

#define TIMEOUT 3

#endif

**common.h:**

#ifndef COMMON\_H

#define COMMON\_H

#define FALSE 0

#define TRUE 1

#define TRANSMITTER 0

#define RECEIVER 1

#endif