U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Redes de Computadores

1º trabalho laboratorial - Ligação de dados

2018/2019

Ana Rita Norinho Pinto | up201606003
Joana Maria Cerqueira da Silva | up201208979
Juliana Maria Cruz Marques | up201605568
Maria Helena Sampaio de Mendonca Montenegro e Almeida | up201604184

Índice

1. Sumário

No âmbito da unidade curricular de Redes de Computadores, foi proposto desenvolver um protocolo de ligação de dados, através de uma porta de série assíncrona, RS-232, entre dois computadores. Para verificar o correto funcionamento do protocolo estabelecido, foi criada uma aplicação simples de teste de transferência de ficheiros.

Os resultados obtidos na conclusão do projeto foram os esperados, uma vez que a transferência de dados é realizada como pretendido, independentemente das situações de teste a que é sujeita.

2. Introdução

O trabalho realizado ao longo das aulas laboratoriais teve como objetivo a implementação de um protocolo de ligação de dados de forma a fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por uma porta de série.

Para isto, foi necessário desenvolver funções de criação e sincronização de tramas (framing), estabelecimento e terminação da ligação, numeração de tramas, confirmação de receção de uma trama sem erros e na sequência correta, controlo de erros e de fluxo.

Para além disso, tivemos que testar o protocolo com uma aplicação simples de transferência de ficheiros, na especificação da aplicação de teste era pedido o suporte de dois tipos de pacotes enviados pelo emissor: o de dados e o de controlo.

Nos pacotes de controlo destacam-se o pacote *start*, sinalizando o início da transmissão; e o pacote *end*, sinalizando o fim da transmissão. Já os pacotes de dados iriam conter partes do ficheiro a transmitir.

Quanto ao relatório, o seu objetivo é expor e explicar toda a componente teórica presente neste primeiro trabalho, tendo a seguinte estrutura:

- **Arquitetura** - Exposição dos blocos funcionais e interfaces.
- **Estrutura do código** - Exposição das APIs, principais estruturas de dados, principais funções e a sua relação com a arquitetura.
- **Casos de uso principais** - Identificação dos casos de uso principais e sequências de chamada de funções.
- **Protocolo de ligação lógica** - Identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes aspetos.
- **Protocolo de Aplicação** - Identificação dos principais aspetos funcionais descrição da estratégia de implementação dos mesmos.
- **Validação** - Descrição dos testes efetuados com apresentação quantificada dos resultados.
- **Eficiência do protocolo de ligação de dados** - Caracterização da eficiência do protocolo, com recurso a medidas sobre o código desenvolvido.
- **Conclusão** - Síntese da informação apresentada anteriormente e reflexão sobre os objetivos de aprendizagem alcançados.

3. Arquitetura

Este projeto encontra-se dividido em dois blocos funcionais: o emissor e o recetor. Cada um destes blocos inclui a camada de ligação de dados e a camada de aplicação.

A camada de ligação de dados disponibiliza funções genéricas do protocolo, estando nela especificadas as funcionalidades de sincronismo, suporte de ligação, controlo da transferência, de erros e de fluxo, assim como todos os aspetos relacionados com a ligação à porta série.

A camada de aplicação é dependente do bloco de ligação de dados, já que tira proveito das suas propriedades para a implementação das suas funções. A camada de aplicação é responsável pela

transferência dos ficheiros pois é nesta que são executadas as funções de receção e emissão de tramas. Os pacotes de controlo e de dados também são processados e enviados a partir desta camada.

4. Estrutura de código

O código está distribuído por três ficheiros: writer.c, reader.c e alarm.c. O primeiro é responsável pela funções do emissor enquanto que o segundo encarrega-se das do recetor.

4.1. Emissor

Funções da camada de ligação:

- **llopen -** envia a trama de supervisão *SET* e recebe a trama de supervisão *UA*.
- **llwrite -** realiza stuffing das tramas de informação e envia-as ao recetor.
- **llclose -** envia trama de supervisão *DISC*, recebe-a e envia trama *UA*.

Funções da camada de aplicação:

- **main -** função principal da camada de ligação uma vez que é a partir desta que o todo funcionamento do programa se desenrola.
- **create_start -** cria o pacote de controlo *START* e invoca o llwrite para o enviar ao recetor.
- **create_data -** divide o ficheiro em pacotes para serem enviados ao recetor.
- **create_end -** cria o pacote de controlo *END* e invoca o llwrite para o enviar ao recetor.

Variáveis globais:

- **set[5] -** trama de supervisão *SET.*
- **trama -** valor *A* do cabeçalho que alterna entre 0x00 e 0x40, inicializada a 0.
- **file[200] -** nome do ficheiro a enviar.
- **n -** número de pacotes em que o ficheiro é dividido.

Macros importantes:

- **BAUDRATE -** capacidade de ligação de dados.
- **MAX -** número máximo de retransmissões.
- **DATA -** campo de controlo do pacote de dados.
- **FLAG -** flag colocada no início e no fim de cada trama a enviar.
- **RR0 -** um dos valores do campo de controlo quando o recetor está pronto para receber informação.
- **RR1 -** um dos valores do campo de controlo quando o recetor está pronto para receber informação.
- **REJ0 -** um dos valores do campo de controlo quando o recetor não está pronto para receber informação.
- **REJ1 -** um dos valores do campo de controlo quando o recetor não está pronto para receber informação.
- **START -** campo de controlo do pacote *START.*
- **END -** campo de controlo do pacote *END.*
- **DISC -** campo de controlo da trama de supervisão enviada no llclose.
- **FILE_LENGTH -** campo T1 do pacote *START.*
- **FILE_NAME -** campo T2 do pacote *START.*

4.2. Recetor

Funções principais da camada de ligação:

- **llopen** – lê trama de controlo *SET* e envia a trama *UA*.
- **llread** – lê tramas I e faz destuffing.
- **llclose** – lê trama de controlo *DISC*, envia *DISC* de volta e recebe *UA*.

Funções principais da camada de aplicação:

- **main** – base da camada de aplicação pois é esta que controla todo o processo que ocorre nesta camada e que faz as chamadas às funções da camada de ligação.
- **save_data –** função principal da transferência de dados que chama llread() e, de acordo com as tramas recebidas, chama as funções seguintes, que as analisarão.
- **analyze_start -** função que cria o ficheiro de acordo com a informação recebida.
- **analyze_data -** função que copia a informação recebida para o ficheiro.

Variáveis globais:

- **Id_trama -** identificador da trama que é suposto receber no llread.
- **STOP -** variável para sair do ciclo de leitura.

Macros importantes:

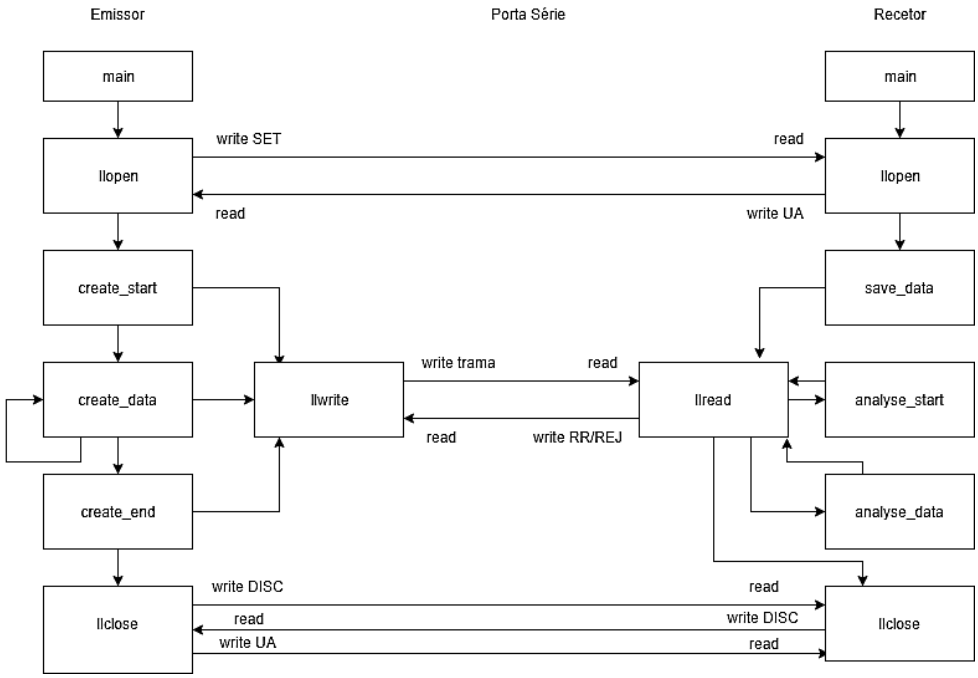
- **BAUDRATE –** Capacidade de ligação**.**
- **RR0 –** confirmação positiva enviada pelo recetor.
- **RR1 –** confirmação positiva enviada pelo recetor.
- **REJ0 –** confirmação negativa enviada pelo recetor.
- **REJ1 –** confirmação negativa enviada pelo recetor.
- **False –** valor booleano 0.
- **True –** valor booleano 1.

5. Casos de usos principais

Inicialmente, nos dois computadores, é necessário escolher a porta de série que irá ser usada na transferência de dados (por ex: /dev/ttyS0). De seguida, a interface do trabalho permite ao utilizador escolher o ficheiro que pretende enviar, para depois ser iniciada a transferência de dados entre ambos os computadores.

No lado do emissor, escolha do ficheiro é realizada logo após ao programa ser compilado e executado, sendo pedido ao utilizador um nome para o mesmo (por ex: pinguim.gif) e depois realizada a transferência de dados. Quanto ao recetor, basta compilar o programa e executá-lo, colocando a porta de série.

Segue-se um gráfico que explica a sequência de chamadas das funções:



A chamada de funções na transferência de dados é dada pela seguinte sequência:

- Emissor escolhe qual o ficheiro a enviar.
- É estabelecida a ligação entre o emissor e o recetor, através da função llopen.
- Criação do pacote de controlo start e envio ao recetor, que cria o ficheiro.
- Divisão do ficheiro em fragmentos e criação do pacote de dados.
- Transmissão de dados através da função llwrite.
- Recetor recebe os dados e envia resposta ao emissor se a transferência foi bem-sucedida.
- Recetor guarda os dados no ficheiro criado.
- Emissor recebe a resposta do recetor, se a transferência não tiver sido bem-sucedida ou caso não tiver chegado nenhuma resposta ao fim de três segundos, volta a retransmitir os dados até a um máximo de três vezes.
- Criação e envio do pacote de controlo end no emissor, que é recebido no recetor e termina a parte de transmissão de dados, fechando o ficheiro criado.
- Terminação da ligação através da função llclose.

6. Protocolo de ligação lógica

A camada de aplicação depende da camada de ligação de dados, implementada neste projeto. A camada de ligação de dados é responsável pelas seguintes funcionalidades:

- Estabelecer e terminar uma ligação através da porta de série, bem como escrever e ler mensagens da mesma;
- Criar e enviar mensagens através da porta de série;
- Receber mensagens através da porta de série;
- Colocar *stuffing* e retirá-lo de pacotes recebidos da camada superior (camada da aplicação).

6.1. LLOPEN

A função ***llopen*** é responsável por estabelecer uma ligação através da porta de série entre o emissor e o recetor.

Quando o emissor invoca esta função, a trama de controlo ***SET*** é enviada através da porta de série e é ativado o alarme que é posteriormente desativado ao receber a resposta do recetor, trama UA.

Se, entretanto, o emissor não receber resposta do recetor (UA) dentro de um tempo pré-definido, *time-out*,, é feita uma nova tentativa e a trama ***SET*** é reenviada. Este ciclo repete-se até o número de tentativas ser ultrapassado, MAX vezes, caso em que a ligação é abortada, ou até a trama ***UA*** ser recebida como resposta do envio do comando ***SET***, caso em que a ligação foi corretamente estabelecida.

Quando o recetor invoca esta função, espera pela receção do comando ***SET***, e quando o recebe, envia o comando de resposta ***UA*** e a ligação é estabelecida.

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

6.2. LLWRITE

Esta função é realizada no emissor e é responsável pelo envio e stuffing das tramas.

Inicialmente, é realizado stuffing por toda a informação a transmitir. Após concluída esta atividade, são acrescentados o cabeçalho e a flag final à mensagem.

De seguida a informação é enviada através da porta de série ao recetor e o alarme é acionado.

O recetor envia uma mensagem de volta e esta é analisada: se tudo correr como esperado, o alarme é desativado e a execução de *LLWRITE* termina; caso contrário, a mensagem é reenviada até um número *MAX* de vezes - se ao fim de *MAX* a mensagem recebida não for esperada a execução do programa termina, dando *time-out*.

6.3. LLREAD

A função ***llread*** é responsável por ler, byte a byte, a trama recebida pela porta série.

Após a leitura, começa por analisar o byte correspondente ao BCC1, retornando um valor de erro caso este esteja errado. Segue a retirar o cabeçalho da trama e a retirar-lhe a redundância introduzida no emissor pelo processo de *stuffing*. Verifica se o valor do BCC2 está de acordo com os bytes de dados recebidos e envia a resposta RR0/1 ou REJ0/1, estando este valor correto ou incorreto, respetivamente. Guardam-se os dados no apontador *buf2*.

Foi utilizada a variável global *id_trama*, para verificar se a trama recebida é a esperada. Caso não o seja, assume-se que ocorreu um duplicado, visto que cada nova trama é enviada no emissor após receber confirmação. Deste modo, é enviado a mensagem RR0/1, no entanto retorna-se com o valor -1, que resulta na trama ser ignorada.

6.4. LLCLOSE

A função ***llclose*** é responsável por terminar a ligação entre o recetor e o emissor através da porta de série. Recebe como parâmetro o identificador da ligação de dados (fd) e retorna um número inteiro: valor positivo em caso de sucesso e negativo em caso de erro.

Quando o emissor chama esta função, o comando ***DISC*** (t*rama de supervisão - disconnect)* é enviado pela porta de série, e aguarda a receção do comando ***DISC***, enviado pelo recetor como confirmação, e finalmente envia o comando ***UA (*** trama não numerada - unnumbered acknowledgment)..

O recetor aguarda pelo comando ***DISC***, enviado pelo emissor, quando o receber reenvia-o e aguarda pela receção do comando ***UA***.

7. Protocolo de aplicação

O protocolo de aplicação tem como aspetos principais:

- A construção dos pacotes *START* e *END*, que contêm o nome e o tamanho do ficheiro a enviar.
- A leitura do ficheiro a enviar, no emissor e criação do ficheiro no recetor.
- A divisão do ficheiro a enviar em segmentos, no caso do emissor e a junção de todos os fragmentos recebidos no caso do recetor.
- As funcionalidades acima descritas foram implementadas através das funções que se seguem.

7.1. CREATE_START

Esta função tem como objectivo criar o pacote de controlo *START* e enviá-lo ao recetor através da função llwrite. Recebe como parâmetros o tamanho do ficheiro a ser enviado (file_length), o nome do ficheiro (fileName) e o identificador da ligação de dados (fd).

Inicialmente, é colocada na primeira posição do pacote o campo de controlo do pacote *START*, que, neste caso, é 0x02. De seguida, é codificado cada parâmetro na forma **TLV (Type, Length, Value)**. É colocada na posição seguinte do pacote o octeto T1, que indica que estamos a colocar o tamanho do ficheiro e, depois, é colocado o octeto L1, que se refere ao tamanho em octetos do campo V (valor do parâmetro). Depois, é então calculado o valor de V para cada posição e colocado, respectivamente, na posição do pacote.

Caso o nome do ficheiro passado como parâmetro da função não seja nulo, é codificado na forma TLV, tal como o parâmetro anterior.

Em último lugar, é enviado o pacote de controlo criado para o recetor através da invocação da função **llwrite**.

7.2. CREATE_DATA

Esta função é responsável pela criação de *N* pacotes de dados com um tamanho específico(*K*).

Inicialmente, o ficheiro a enviar é aberto e dividido em segmentos a enviar.Por cada segmento lido, é criado um pacote cuja primeira posição contém o valor *DATA*, a segunda o número de sequência do mesmo e nas terceiras e quartas, respetivamente L2 e L1, o número em octetos do campo de dados, ou seja, do tamanho do segmento lido,no formato L2L1, calculados a partir de ***K* =255*L2+L1.**

Por fim, é chamada a função llwrite de modo a enviar ao recetor o pacote de dados criado.

7.3. CREATE_END

Esta função é responsável pela criação do pacote de controlo *END* e envia-lo, através de llwrite, ao recetor. Este assemelha-se ao pacote de controlo *START*, criado na função **create_start**, sendo a única diferença a primeira posicão que, neste caso, contém o valor *END*.

7.4. SAVE_DATA

Esta é a função da camada da aplicação que acede à camada de ligação de dados ao chamar a função llread. De acordo com o resultado de llread, a função irá escolher o que fazer com a informação recebida do seguinte modo:

Se llread retorna -1, ocorreu um erro ou foi recebido uma trama duplicada pelo que se retorna ao início do ciclo e repete-se a função. Em caso contrário, foi recebida uma trama, passando a ser analisado o primeiro byte da mesma, para definir se é a trama de início, uma trama de dados ou a trama de fim. Para a trama de início, chama a função analyse_start. Para uma trama de dados chama a

função analyse_data. Para a trama de fim, fecha o ficheiro e sai do ciclo de leitura, retornando ao main.

7.5. ANALYSE_START

Esta função está responsável por analisar a trama de início e criar um novo ficheiro de acordo com essa informação. Retorna o descritor do ficheiro para que este possa ser modificado de acordo com os dados recebidos.

7.6. ANALYSE_DATA

Esta função recebe um array com os dados recebidos como argumento e o descritor do ficheiro, estando responsável por analisar e guardar os dados no ficheiro.

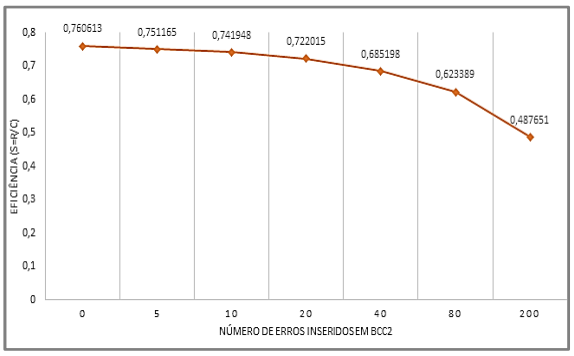
8. Validação

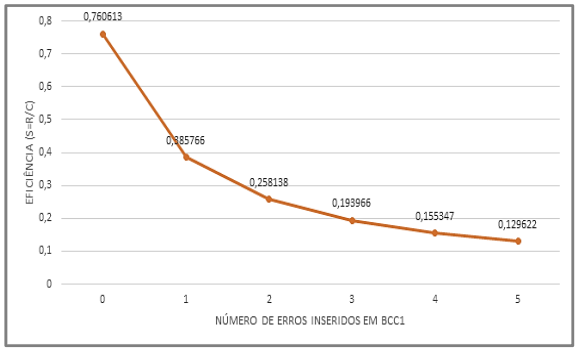
Foram realizados os seguintes testes, todos eles acabados em sucesso:

- Interrupção da ligação de dados durante o envio de informação.
- Simulação de um curto circuito durante o envio de informação.
- Envio de informação de um ficheiro de texto.

9. Eficiência de protocolo de ligação de dados

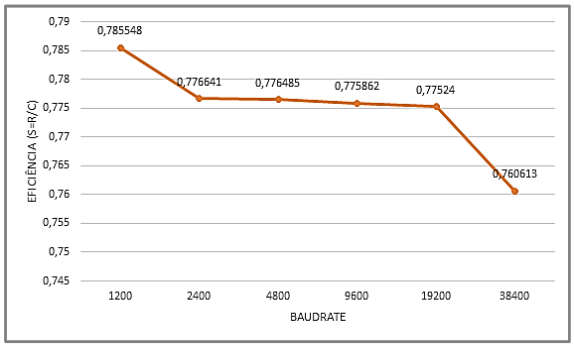
Teste de variação de FER, introduzindo erros no BBC1 e BCC2:





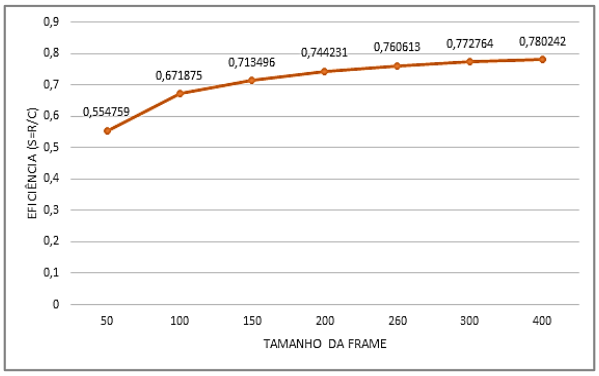
A partir destes gráficos conclui-se que quanto mais erros tiverem as tramas menor é a eficiência do protocolo. Tal deve-se ao facto de o emissor ter que transmitir a mesma trama novamente sempre que se inserem erros. É de notar que a eficiência quando se inserem erros no BCC1 desce mais rápido que no BCC2, o que se explica por quando se encontram erros no BCC1 não é enviada resposta do receptor para o emissor, pelo que o emissor tem que esperar 3 segundos para transmitir novamente a informação.

Teste de variação da capacidade da ligação:



Quanto menor for a capacidade da ligação, maior a eficiência, apesar de demorar mais tempo.

Teste de variação do tamanho das tramas:



Quanto maior for o tamanho da trama a enviar, maior é a eficiência e menor é o tempo que demora a concluir a ação. Isto deve-se ao facto de quanto maior o tamanho da trama menor o número de tramas a enviar.

Mecanismo STOP&WAIT:

O mecanismo STOP&WAIT, em que se baseou este projeto, indica que o emissor deve esperar por uma resposta após a enviar uma trama. A resposta pode ser positiva (ACK como designado no mecanismo ou RR no nosso projeto), que indica que pode ser enviada a seguinte trama, ou negativa (NACK ou REJ no nosso projeto), que indica que a trama deve ser reenviada. Caso uma mensagem seja perdida durante o processo, existe um mecanismo de timeout que procede ao reenvio da trama no emissor. No nosso projeto existe ainda uma diferenciação entre as tramas, 0 ou 1, para ser possível verificar a existência de duplicados.

10.Conclusões

Neste projeto foi desenvolvido com o objetivo de executar uma transmissão de dados entre dois sistemas, utilizando um mecanismo STOP&WAIT onde é enviada uma trama de informação de cada vez. Neste processo, implementamos um programa capaz de transmitir informação com sucesso, mesmo em situações inesperadas como a de interrupção da transmissão, tendo desenvolvido um sistema de verificação de erros apropriado.

O projeto desenvolvido permitiu-nos aprofundar os nossos conhecimentos sobre a linguagem C, para além de nos ajudar a consolidar a matéria dada sobre as responsabilidades de cada camada, nomeadamente a camada de aplicação e a camada de ligação de dados, e as relações entre elas.

11.Anexos

11.1. Reader.c

```c
/*LLOPEN Emissor Processing*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

// MACROS
#define BAUDRATE B38400
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define RR0 0x05
#define RR1 0x85
#define REJ0 0x01
#define REJ1 0x81

// Variaveis globais
int id_trama=0;
void analyze_data(unsigned char * buffer, int size, int fd);
volatile int STOP=FALSE;

// Declarações de funções
int main(int argc, char** argv);
int llopen(int fd);
int llread(int fd, unsigned char* buf2);
int save_data(int fd);
int analyze_start(unsigned char* buffer, int size);
void analyze_data(unsigned char * buffer, int size, int fd);
int llclose(int fd);

/*
* Base da camada de aplicação, é esta que controla todo o processo
* que ocorre nesta camada e que faz as chamadas às funções da camada
* de ligação.
*/
int main(int argc, char** argv)
{
    int fd,c, res, i=0;
    struct termios oldtio,newtio;
    unsigned char buf[255];

    if ( (argc < 2) ||
         ((strcmp("/dev/ttyS0", argv[1])!=0) &&
          (strcmp("/dev/ttyS1", argv[1])!=0) )) {
        printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
        exit(1);
    }

    fd = open(argv[1], O_RDWR | O_NOCTTY );
    if (fd <0) {perror(argv[1]); exit(-1); }

    if ( tcgetattr(fd,&oldtio) == -1) {
      perror("tcgetattr");
      exit(-1);
    }

    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;
    newtio.c_lflag = 0;

    newtio.c_cc[VTIME]    = 1;
```

```
    newtio.c_cc[VMIN]     = 0;

    tcflush(fd, TCIOFLUSH);

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

    if(llopen(fd)==-1) { return -1; }

    // para medição dos tempos
    struct timespec start_t, end_t;
    clock_gettime(CLOCK_REALTIME, &start_t);

    save_data(fd);

    clock_gettime(CLOCK_REALTIME, &end_t);
    printf("time: %f\n", (double) (end_t.tv_sec-start_t.tv_sec + (end_t.tv_nsec-start_t.tv_nsec)/1E9));

    if(llclose(fd)==-1)
        return -1;

    tcsetattr(fd,TCSANOW,&oldtio);

    close(fd);
    return 0;
}

/*
* Lê trama de controlo SET e envia a trama UA.
*/
int llopen(int fd){

    unsigned char SET[5], buf;
    SET[0]  = 0x7E;
    int i=0, res;
    int state = 0;

    while(!STOP)
    {
        res = read(fd, &buf,1);

        if(res!=0)
        {
            switch(state)
            {
            case 0:
                if(buf == 0x7E)
                    state++;
                break;
            case 1:
                if(buf != 0x7E) {
                    state++;
                    i++;
                    SET[i] = buf;
                }
                break;
            case 2:
                if(buf != 0x7E && i >= 4)
                    STOP = TRUE;
                else if(buf != 0x7E){
                    i++;
                    SET[i] = buf;
                }
                else{
```

```
                    i++;
                    SET[i] = buf;
                    STOP = TRUE;
                }
                break;
            }
        }
    }

    //analise
    if(SET[3]!= SET[1]^SET[2] && SET[2]!=0x03)
        return -1;

    //estrutura a enviar UA
    unsigned char UA[5];
    UA[0] = 0x7E;
    UA[1] = 0x03;
    UA[2] = 0x07;
    UA[3] = 0x04;
    UA[4] = 0x7E;
    res = write(fd, UA, sizeof(UA));
    return 0;
}

/*
* Lê tramas I e faz destuffing.
*/
int llread(int fd, unsigned char* buf2){

    unsigned char buf;
    unsigned char trama[512];
    trama[0] = 0x7E;
    int i=0, res;
    int state = 0;
    STOP = FALSE;

    while(!STOP)
    {
        res = read(fd, &buf,1);

        if(res!=0)
        {
            switch(state)
            {
            case 0:
                if(buf == 0x7E)
                    state++;
                break;
            case 1:
                if(buf != 0x7E) {
                    state++;
                    i++;
                    trama[i] = buf;
                }
                break;
            case 2:
                if(buf != 0x7E){
                    i++;
                    trama[i] = buf;
                }
                else{
                    i++;
                    trama[i] = buf;
                    STOP = TRUE;}
```

```
                break;
            }
        }
    }
    printf("N: %d\n", (int) trama[5]);

    unsigned char message[5];
    message[0] = 0x7E;
    message[1] = 0x03;
    message[4] = 0x7E;

    if(trama[3]!= trama[1]^trama[2] && trama[2]!=0x00 && trama[2]!=0x40){

        if(trama[2]==0x00 && id_trama == 0){
            printf("REJ0\n");
            message[2] = REJ0;
            message[3] == message[1]^message[2];
            write(fd, message, 5);
            return -1;
        }
        if(trama[2]==0x00 && id_trama != 0){
            printf("RR1\n");
            message[2] = RR1;
            message[3] == message[1]^message[2];
            write(fd, message, 5);
            return -1;
        }
        if(trama[2]==0x40 && id_trama == 1){
            printf("REJ1\n");
            message[2] = REJ1;
            message[3] == message[1]^message[2];
            write(fd, message, 5);
            return -1;
        }
        if(trama[2]==0x40 && id_trama != 1){
            printf("RR0\n");
            message[2] = RR0;
            message[3] == message[1]^message[2];
            write(fd, message, 5);
            return -1;
        }
        return -1;
    }

    if(trama[2]==0x00 && id_trama != 0){
        printf("RR1 duplicate\n");
        message[2] = RR1;
        message[3] == message[1]^message[2];
        write(fd, message, 5);
        return -1;
    }
    if(trama[2]==0x40 && id_trama != 1){
        printf("RR0 duplicate\n");
        message[2] = RR0;
        message[3] == message[1]^message[2];
        write(fd, message, 5);
        return -1;
    }

    int n =0;
    unsigned char bcc2;
    int j = 4;

    while(j < i-1)
    {
        if(trama[j] == 0x7D && trama[j+1] == 0x5E)  /*ciclo para apgar 0x5E e colocar 0x7D=0x7E */
        {
```

```
            buf2[n] = 0x7E;
            j++;
        }
        else if(trama[j] == 0x7D && trama[j+1] == 0x5D) /*ciclo para apgar 0x5D*/
        {
            buf2[n] = 0x7D;
            j++;
        }
        else {
            buf2[n] = trama[j];
        }
        if(n==0)
            bcc2=buf2[n];
        else
            bcc2=bcc2^buf2[n];
        n++;
        j++;
    }


    if(bcc2 != trama[i-1] &&  id_trama == 0)
    {
            printf("REJ0 entered\n");
            message[2] = REJ0;
            message[3] == message[1]^message[2];
            write(fd, message, 5);
            return -1;
    }

    else if(bcc2 != trama[i-1] && id_trama == 1)
    {
            printf("REJ1 entered\n");
            message[2] = 0x81;
            message[3] == message[1]^message[2];
            write(fd, message, 5);
            return -1;
    }

    if(trama[2]==0x00 && id_trama == 0){
        printf("RR1\n");
        message[2] = RR1;
        message[3] == message[1]^message[2];
        write(fd, message, 5);
        return n-1;
    }

    if(trama[2]==0x40 && id_trama == 1){
        printf("RR0\n");
        message[2] = RR0;
        message[3] == message[1]^message[2];
        write(fd, message, 5);
        return n-1;
    }

}

/*
* Função principal da transferência de dados que chama llread() e,
* de acordo com as tramas recebidas, chama as funções seguintes, que as analisarão.
*/
int save_data(int fd){

    unsigned char buffer[512];
    int stop2 = 0, size, fd1;
    while(!stop2)
    {
```

```
        size = llread(fd, buffer);

        if(size <= 0) continue;

        if(id_trama == 0)
            id_trama=1;

        else if (id_trama == 1)
            id_trama = 0;

        if(buffer[0] == 0x02)
            fd1=analyze_start(buffer, size);

        else if(buffer[0]==0x01)

            analyze_data(buffer, size, fd1);


        else if(buffer[0]==0x03)
        {
            close(fd1);
            stop2=1;
        }
    }

    return 0;
}

/* Função que cria o ficheiro de acordo com a informação recebida.*/
int analyze_start(unsigned char* buffer, int size) {

    int l1, l2, i=1;

    if(buffer[1] == 0x00)
    {
        l1 =  (int) buffer[2];

        if(buffer[3+l1]==0x01)
        {
            //l2 = tamanho do nome do ficheiro
            l2 = (int) buffer[4+l1];
        }
    }
    unsigned char nome[l2+1];

    while(i <= l2)
    {
        nome[i-1]=buffer[4+l1+i];
        i++;
    }

    nome[l2] = '\0';
    int fd= open(nome, O_CREAT | O_WRONLY |O_APPEND);
    return fd;
}

/* Função que copia a informação recebida para o ficheiro.*/
void analyze_data(unsigned char * buffer, int size, int fd){

    unsigned int l2=(unsigned int) buffer[2];
    unsigned int l1=(unsigned int) buffer[3];

    int k = l2*256+l1;
    unsigned char toWrite[k];
    int i=0;

```

```
    while(i < k)
    {
        toWrite[i] = buffer[i+4];
        i++;
    }
    write(fd, toWrite, k);
}

/*
* Lê trama de controlo DISC, envia DISC de volta e recebe UA.
*/
int llclose(int fd){

    unsigned char DISC[5], buf;
    DISC[0]  = 0x7E;
    int i=0, res;
    int state = 0;
    STOP = FALSE;

   while(!STOP)
   {
        res = read(fd, &buf,1);

        if(res!=0)
        {

            switch(state)
            {
            case 0:
                if(buf == 0x7E)
                    state++;
                break;
            case 1:
                if(buf != 0x7E) {
                    state++;
                    i++;
                    DISC[i] = buf;
                }
                break;
            case 2:
                if(buf != 0x7E && i >= 4)
                    STOP = TRUE;
                else if(buf != 0x7E){
                    i++;
                    DISC[i] = buf;
                }
                else{ i++;
                    DISC[i] = buf;
                    STOP = TRUE;
                }
                break;
            }
        }
    }


    if(DISC[3]!= DISC[1]^DISC[2] && DISC[2]!=0x0B)
        return -1;
    res = write(fd, DISC, 5);

    unsigned char UA[5];

    while(!STOP)
    {
        res = read(fd, &buf,1);

```

```
462         {
463             switch(state)
464             {
465             case 0:
466                 if(buf == 0x7E) state++;
467                 break;
468             case 1:
469                 if(buf != 0x7E) {
470                     state++;
471                     i++;
472                     UA[i] = buf;
473                 }
474                 break;
475             case 2:
476                 if(buf != 0x7E && i >= 4)
477                     STOP = TRUE;
478                 else if(buf != 0x7E){
479                     i++;
480                     UA[i] = buf;
481                 }
482                 else{
483                     i++;
484                     UA[i] = buf;
485                     STOP = TRUE;
486                 }
487                 break;
488             }
489         }
490     }
491     if(UA[3]!= UA[1]^UA[2] && UA[2]!=0x0B)
492         return -1;
493     return 0;
494 }
```

11.2. Writer.c

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <math.h>
#include "alarme.c"
#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define MAX 3
#define DATA 0x01
#define FLAG 0x7E
#define RR0 0x05
#define RR1 0x85
#define REJ0 0x01
#define REJ1 0x81
#define START 0x02
#define END 0x03
#define DISC 0x0B
#define FILE_LENGTH 0x00
#define FILE_NAME 0x01


int llopen(int fd);
int create_start(int file_length,unsigned char *fileName,int fd);
int create_data(int fd);
int llwrite(int fd, unsigned char* set1,int size);
int create_end(int file_length,unsigned char *fileName,int fd);
void atende();
volatile int STOP=FALSE;
int received=0;
unsigned char set[5];
unsigned char trama=0x00;
unsigned char file[200];
int n = 1; //sequenceNumber
int k  = 260;

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

    int fd,c, res;
    struct termios oldtio,newtio;
    (void) signal(SIGALRM,atende);
    set[0]= FLAG;
    set[1]= 0x03; //A
    set[2]= 0x03;//C, define a trama que se esta a usar, 0x03 - transmicao emissor -> recetor
    set[3]= set[1]^set[2];
    set[4]=FLAG;

    char buf[255];
    int i, sum = 0, speed = 0;

    if ( (argc < 2) ||
         ((strcmp("/dev/ttyS0", argv[1])!=0) &&
          (strcmp("/dev/ttyS1", argv[1])!=0) )) {
      printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
      exit(1);
    }

```

```
67
68      /*
69        Open serial port device for reading and writing and not as controlling tty
70        because we don't want to get killed if linenoise sends CTRL-C.
71      */
72
73        fd = open(argv[1], O_RDWR | O_NOCTTY );
74        if (fd <0) {perror(argv[1]); exit(-1); }
75
76        if ( tcgetattr(fd,&oldtio) == -1) { /* save current port settings */
77          perror("tcgetattr");
78          exit(-1);
79        }
80
81        bzero(&newtio, sizeof(newtio));
82        newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
83        newtio.c_iflag = IGNPAR;
84        newtio.c_oflag = 0;
85
86        /* set input mode (non-canonical, no echo,...) */
87        newtio.c_lflag = 0;
88
89        newtio.c_cc[VTIME]    = 1;   /* inter-character timer unused */
90        newtio.c_cc[VMIN]     = 0;   /* blocking read until 5 chars received */
91
92
93
94      /*
95        VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a
96        leitura do(s) pr?ximo(s) caracter(es)
97      */
98
99        tcflush(fd, TCIOFLUSH);
100
101         if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
102           perror("tcsetattr");
103           exit(-1);
104         }
105
106
107       /*printf("Please enter the name of the file: ");
108       scanf("%s",file);*/
109
110       strcpy(file,"pinguim.gif");
111
112       struct stat st;
113       int file_size;
114       stat(file, &st);
115       file_size = st.st_size;
116
117       llopen(fd);
118       create_start(file_size, file,fd);
119       create_data(fd);
120       create_end(file_size,file,fd);
121       llclose(fd);
122       close(fd);
123
124       return 0;
125     }
126     int create_start(int file_length,unsigned char *fileName,int fd){
127
128       unsigned char pack_start[256];
129       pack_start[0]=START;
130       int i = 0;
131       unsigned char v1[256];
132       do {
133         v1[0]=(char)(file_length/pow(256.0,(double)i)) % 256;
```

```
134         i++;
135       }while(v1[i]!=0);
136
137       pack_start[1]= FILE_LENGTH; //t1
138       pack_start[2]=i; //l1
139
140       int j;
141       for (j=3;j<i+3;j++){
142         pack_start[j]=v1[j-3];
143       }
144       i+=3;
145       if (fileName != NULL){
146         pack_start[i]=FILE_NAME; //t2
147         int str_length=strlen(fileName);
148
149         i++;
150         pack_start[i]=str_length; //l2
151         i++;
152
153         for (j = 0;j<str_length;j++){
154           pack_start[i]=fileName[j]; //v2
155           i++;
156         }
157       }
158
159         int res = llwrite(fd,pack_start,i+1);
160         trama=0x40;
161     }
162
163
164     int create_data(int fd){
165       unsigned char pack_data[k];
166       int i =0;
167       unsigned char dest[k];
168      unsigned char filename[20];
169
170       strcpy(filename,file);
171       int open_file = open(filename,O_RDONLY);
172       int res;
173
174
175      while((res=read(open_file, dest,k))>0){
176
177       pack_data[i]=DATA;
178       i++;
179       pack_data[i] = (unsigned int) n;
180       i++;
181       pack_data[i] = (unsigned int) res / 256; //l2
182       i++;
183       pack_data[i] = (unsigned int) res % 256; //l1
184       i++;
185
186       int j=0;
187       for(j;j<res;j++){
188         pack_data[i] = dest[j];
189         i++;
190       }
191
192       int res = llwrite(fd,pack_data,i+1);
193
194       n++;
195       printf("Sequence Number N: %d\n",n );
196       i=0;
197       if ((n%2)==0)
198         trama=0x00;
199       else
200         trama =0x40;
201
```

```
202     }
203
204   }
205   int create_end(int file_length,unsigned char *fileName,int fd){
206     unsigned char pack_end[256];
207     pack_end[0]=END;
208     int i = 0;
209     unsigned char v1[256];
210     do {
211           v1[0]=(int)(file_length/pow(256.0,(double)i)) % 256;
212           i++;
213     }while(v1[i]!=0);
214
215     pack_end[1]= FILE_LENGTH;
216     pack_end[2]=i;
217
218     int j;
219     for (j=3;j<i;j++){
220       pack_end[j]=v1[j-3];
221     }
222
223     i+=4;
224     if (fileName != NULL){
225       pack_end[i]=FILE_NAME;
226       int str_length=strlen(fileName);
227       i++;
228       pack_end[i]=str_length;
229       i++;
230       for(j = 0;j<str_length;j++){
231         pack_end[i]=fileName[j];
232         i++;
233       }
234     }
235
236     int res = llwrite(fd,pack_end,i+1);
237   }
238
239   int llopen(int fd){
240     int res;
241     int i=0;
242     unsigned char buf[5];
243     conta_zero();
244
245     while(conta<MAX && !received){
246       desativa_alarme();
247
248       res=write(fd,set,sizeof(set));
249
250       alarm(3);
251       while(!flag && !received){
252
253         res=read(fd,buf+i,1);
254
255         if (res!=0)
256           if(i<5 && res!=0) {
257           switch(buf[i]){
258             case 0x7E:
259             if (i==0 || i==4)
260               i++;
261             break;
262             default:
263             if (i!=0 && i!=4)
264               i++;
265
266           }
267       }
268         if (i==5){
269         if (buf[2]==0x07 && buf[3]==buf[1]^buf[2]) {
```

```
270                 received=1;
271                 desativa_alarme();
272                 printf("Llopen successful!\n");
273                 }
274               }
275             }
276   }
277   }
278
279
280 int llwrite(int fd,unsigned char* set1,int size){
281
282     unsigned char buf1[512];
283     unsigned char buf2[512];
284     unsigned char buf3[512];
285     unsigned char bcc2;
286     unsigned char confirmation;
287     int res=1;
288     int i,j;
289     i=0;
290     j=0;
291     flag=1;
292     for(; i<size;i++){
293       buf1[i]=set1[i];
294
295
296         if (buf1[i]==0x7E){
297           buf2[j]=0x7D;
298           buf2[j+1]=0x5E;
299           j++;
300
301         }
302         else if(buf1[i]==0x7D){
303           buf2[j]=0x7D;
303           buf2[j]=0x7D;
304           buf2[j+1]=0x5D;
305           j++;
306         }
307         else
308           {
309             buf2[j]=buf1[i];
310           }
311           j++;
312           if (i==0){
313             bcc2=buf1[i];
314           }else
315         bcc2=bcc2^buf1[i];
316
317
318     }
319
320     int aux2=i;
321     buf2[j]=bcc2;
322     buf3[0]=FLAG;
323     buf3[1]=0x03;
324     buf3[2]=trama;
325     buf3[3]=buf3[1]^buf3[2];
326     i=4;
327
328   if (trama ==0x00)
329       confirmation=RR1;
330   else
331     confirmation=RR0;
332
333     while(i<=j+4){
334       buf3[i]=buf2[i-4];
335       i++;
336     }
```

```
338     buf3[i]=FLAG;
339     conta_zero();
340     received=0;
341     int aux = i+1;
342
343     while(conta <= MAX && !received){
344     desativa_alarme();
345
346     res=write(fd,buf3,aux);
347     i=0;
348
349     unsigned char buf[512];
350     alarm(3);
351     flag=0;
352     while(!flag && !received){
353     res=read(fd,buf+i,1);
354
355       if(res!=0) {
356       switch(buf[i]){
357         case 0x7E:
358         if (i==0 || i==4)
359           i++;
360         break;
361         default:
362         if (i!=0 && i!=4)
363           i++;
364   }
365       if (i==5){
366         if (buf[2] == confirmation && buf[3]==buf[1]^buf[2]) {
367           received=1;
368           desativa_alarme();
369           }
370         else if(buf[2]== REJ0 && trama == 0x00){
371          printf("Restransmiting - received REJ0\n");
372          return llwrite(fd,set1,size);
373         }
374         else if(buf[2]== REJ1 && trama == 0x40){
375          printf("Restransmiting - received REJ1\n");
376         return llwrite(fd,set1,size);
377         }
378     }
379   }
380   }
381   }
382
383   if(received == 0){
384     printf("timeout!\n");
385     exit(0);
386   }
387   return 0;
388   }
389
390   int llclose(int fd){
391     int res;
392     int i=0;
393     unsigned char buf[5];
394     conta_zero();
395     set[2]=DISC;
396     set[3] = set[1] ^ set[2];
397     received = 0;
398
399     while(conta<MAX && !received){
400       desativa_alarme();
401
402       res=write(fd,set,sizeof(set));
403
404       alarm(3);
405       while(!flag && !received){
```

```
406
407       res=read(fd,buf+i,1);
408
409     if (res!=0)
410       if(i<5 && res!=0) {
411       switch(buf[i]){
412         case 0x7E:
413         if (i==0 || i==4)
414           i++;
415         break;
416         default:
417         if (i!=0 && i!=4)
418           i++;
419
420       }
421     }
422       if (i==5){
423       if (buf[2]==DISC && buf[3]==buf[1]^buf[2]) {
424         received=1;
425         set[2]=0x07;
426         set[3] = set[1] ^ set[2];
427         res=write(fd,set,sizeof(set));
428         desativa_alarme();
429         }
430       }
431     }
432
433   printf("Llclose successful!\n");
434   }
435   }
436
```

11.3. Alarm.c

```
1   #include <unistd.h>
2   #include <signal.h>
3   #include <stdio.h>
4
5   int flag=1, conta=1;
6
7   void atende()                   // atende alarme
8   {
9       printf("alarme # %d\n", conta+1);
10      flag=1;
11      conta++;
12  }
13
14
15  void desativa_alarme(){
16      flag=0;
17      alarm(0);
18
19  }
20  void conta_zero(){
21  conta=0;
22  }
```