MIEIC - 2015/2016

Redes de Computadores

Protocolo de Ligação de Dados

Data de Entrega: 6/11/15

Professora: Ana Aguiar

Turma: 2

Trabalho realizado por:

Luís Carvalho - up201303030

João Cepa - ei12067

Vitor Esteves - up201303104

Sumário

O trabalho laboratorial da disciplina consistiu no desenvolvimento de uma aplicação com a finalidade de ser capaz de transferir um ficheiro de imagem entre dois computadores utilizando uma porta série.

A proposta de trabalho revelou-se inicialmente de difícil compreensão. No entanto, há medida que as diferentes camadas se iam construindo, e que mais testes eram realizados, o grupo reagiu bem as dificuldades e todas foram facilmente ultrapassadas.

Introdução

Com este trabalho pretendemos criar uma aplicação que fosse capaz de transferir ficheiros entre dois computadores utilizando uma porta de série. O relatório produzido consiste assim, numa espécie de tutorial de apresentação, com o objetivo de dar a conhecer todas as funcionalidades da mesma. Ao longo das diferentes secções apresentadas serão abordados pontos-chave da implementação, acompanhados com imagens de código e explicações textuais de modo a facilitar a utilização do programa e ainda, se necessário, proceder a alterações.

É inicialmente descrita a **arquitetura** do programa, onde serão explorados as duas camadas do projeto: aplicação e protocolo de ligação de dados. Em seguida, é brevemente apresentada a **estrutura do código** onde se procederá à apresentação das principais funções de cada classe. São ainda explicados os principais aspetos funcionais tanto do **protocolo de ligação lógica** como do **protocolo de aplicação** e a estratégia utilizada na implementação dos mesmos.

Por fim, serão descritos os **testes** efetuados ao programa e enumerados os **elementos de valorização** implementados.

Arquitetura

Existem duas estruturas base no nosso programa: a nível superior e a um nível mais baixo, respectivamente, a estrutura da aplicação e da ligação de dados.

No caso da camada da aplicação, é tratada a leitura da imagem a transferir, a sua divisão em partes e a do pacote de dados utilizado para a transportar.

A camada de ligação de dados encarrega-se do byte stuffing e do envio de dados após a ligação à porta de série que é feita no início, ao executar a aplicação e é composta pelas funções, llopen(), llclose(), sendSET(), sendUA(), linkwrite(), linkread(), senderDISC() e receiverDISC(), enquanto que a camada da aplicação é composta pelas restantes.

Estrutura do Código

A implementação foi dividida em diferentes classes: app.c, data_transfer.c, establishment.c, termination.c e main.c.

app.c :

- **llopen**(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar) -> Abertura da ligação para leitura e escrita de dados.

- **llclose**(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar) -> Fecho da ligação de dados (trama DISC).

- **llwrite**(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar)

- **llread** (Settings* structDados)

data_transfer.c :

- **linkwrite**(unsigned char* data, Settings* structDados, int datasize, int Ns, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar) -> Chamada apenas no lado do emissor, recebe de llwrite() os pacotes a transmitir e coloca-os dentro de tramas, tal como especificado.

- **linkread**(unsigned char* dataPackage, Settings* structDados, int duplicate) -> Chamada apenas no lado do recetor, lê da porta série as tramas transmitidas pelo emissor, separa os pacotes de dados e passa-os para llread() para o seu posterior processamento.

establishment.c :

- **sendSET**(int fd, Settings* structDados, int estado, int tentativaEnvio, int podeEnviar) -> Chamada apenas no lado do emissor. Envia uma trama SET

- **sendUA**(int fd, Settings* structDados, int estado) -> Envia uma trama UA

termination.c :

- **senderDISC**(unsigned char* DISC, Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar) -> Envia um DISC. Chamada apenas no lado do emissor.

- **receiverDISC**(unsigned char* DISC, Settings* structDados, volatile int estado) -> Envia um DISC. Chamada apenas no lado do recetor.

Casos de uso principais

Inicialmente, é pedido ao utilizador para introduzir uma série de configurações à escolha através do uso da interface:

1º. É efetuada a escolha (em ambos os computadores) de qual será o computador *Receiver* e qual será o *Sender*.

2º. Em seguida, o utilizador deve selecionar qual a porta que está a usar, e no caso do emissor, escolher o nome do ficheiro a enviar.

3º . De entre as várias opções de valores é permitido escolher qual o *baudrate* a utilizar.

4º. No caso do receptor, apenas falta escolher o timeout e o número de tentativas que o programa irá usar para enviar o ficheiro. No caso do emissor, para além das opções acima, é ainda necessário escolher o tamanho do ficheiro a enviar.

A partir daqui, o programa chama a função llopen onde se envia uma trama SET ou UA, dependendo se é emissor ou recetor, respetivamente, e que retorna um descritor para a porta série. Neste ponto, ambos os programas (computador emissor vs. computador recetor) estarão em funções diferentes. Enquanto o emissor abre o ficheiro que quer enviar de modo a obter o tamanho do ficheiro, o recetor permanece em standby até receber um pacote por parte do outro computador. Após terminar a verificação do tamanho do ficheiro, o emissor começa a preparar um pacote de dados, na função llwrite, que, por sua vez, irá utilizar a função linkwrite para escrever na porta série a trama que foi preparada. Após enviar, o emissor ficará a espera até receber uma resposta por parte do recetor ou até o timeout ter passado. Caso tal aconteça, o emissor irá enviar novamente a trama e ficará novamente à espera. De modo a não ficar em loop infinitamente, o emissor apenas irá repetir este ciclo de envio-espera, um numero pré-determinado de vezes. Por outro lado, quando o recetor recebe algo através da função linkread, vai primeiro verificar possíveis erros que a trama possa conter através de alguns campos de controlo. Caso isso não se verifique, o linkread retorna com sucesso, o que fará com que a função llread envie uma resposta positiva RR a pedir um novo pacote. Caso a verificação considere que existe algum erro na trama, retorna de forma errada, o que fará com que o llread envie um REJ, ou seja, um alerta para o emissor voltar a transmitir o mesmo pacote. Após terminar de enviar corretamente todos os pacotes, o emissor envia um DISC e recebe outro por parte do recetor. Para fechar a porta, o emissor envia uma última trama UA para o recetor.

Protocolo de Ligação Lógica

O protocolo tem como objetivo organizar a informação do ficheiro. No entanto, contrariamente às tramas de supervisão, as usadas neste protocolo enviam informações sobre o ficheiro. No código seguinte, é possível ver que todas as tramas possuem quatro campos comuns: a FLAG, no primeiro índice (0) e uma última no último índice, um campo de endereço (A), um NS na segunda posição, e ainda um campo de controlo BCC que é posto no terceiro índice da trama.

```
unsigned char FRAME_I[6 + datasize];
        FRAME_I[0] = FLAG;
        FRAME_I[1] = A;
        FRAME_I[2] = Ns == 0 ? 0x00 : 0x20;
        FRAME_I[3] = FRAME_I[1] ^ FRAME_I[2];
unsigned char BCC2 = 0x00;
```

É agora importante guardar o conjunto de bytes a enviar relativos ao ficheiro e, como tal, continuamos assim a preencher ainda mais índices da trama. Inicialmente é feito um ciclo que tem como objetivo calcular o BCC2, com a finalidade de confirmar, mais tarde, se a trama chegou corretamente ao receptor. Adiciona-se ainda a FRAME_I, um pacote de dados temporário, os dados que se encontram em data (array recebido como parâmetro na função). No final, transferem-se os dados do pacote temporário para o pacote final, FRAME_I_FINAL, e aplica-se o método de byte stuffing, de modo a que o programa ignore as flags que encontrar que não estejam no primeiro índice nem no último.

```
for (i = 0; i < datasize; i++) {
        FRAME_I[4 + i] = data[i];
        BCC2 ^= data[i];
        if (FRAME_I[4 + i] == 0x7e || FRAME_I[4 + i] == 0x7d) aSubstituir++;
}
FRAME_I[4 + datasize] = BCC2;
if (FRAME_I[4 + datasize] == 0x7e || FRAME_I[4 + datasize] == 0x7d) aSubstituir++;
FRAME_I[5 + datasize] = FLAG;

unsigned char FRAME_I_FINAL[6 + datasize + aSubstituir];
int j;
for (i = 0, j = 0; i < 6 + datasize + aSubstituir; i++, j++) {
        if (j < 4 || j == 6 + datasize - 1) FRAME_I_FINAL[i] = FRAME_I[j];
        else if (FRAME_I[j] == 0x7e || FRAME_I[j] == 0x7d) {
                FRAME_I_FINAL[i++] = 0x7d;
                FRAME_I_FINAL[i] = 0x20 ^ FRAME_I[j];
        }
        else FRAME_I_FINAL[i] = FRAME_I[j];
}
```

Protocolo de Aplicação

Este protocolo destina-se ao processamento da imagem. No lado do emissor, é lido sequencialmente o ficheiro a enviar e dividido em bytes, que posteriormente são colocados dentro de tramas I. No receptor, são escritos os bytes recebidos num novo ficheiro de imagem com o mesmo nome do original.

Na parte do emissor é criado um array onde se coloca toda a informação do primeiro pacote de controlo, ou seja, do inicio da transmissão. A informação corresponde ao tamanho e nome do ficheiro e ao comprimento máximo do campo de dados das futuras tramas I.

```
int llwrite(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile
int podeEnviar) {
        int Ns = 0;
        unsigned int i = 0;

        unsigned char CTRL_START[11 + strlen(structDados->fileName) + 1];
        CTRL_START[i++] = 0x01; // Indica início
        CTRL_START[i++] = 0x00; // A enviar tamanho do ficheiro
        CTRL_START[i++] = 0x02; // 2 bytes
        CTRL_START[i++] = (unsigned char)((structDados->filesize & 0xff00) >> 8); // Tamanho do
```

```
ficheiro
        CTRL_START[i++] = (unsigned char)(structDados->filesize & 0xff);
        CTRL_START[i++] = 0x01; // A enviar nome do ficheiro
        CTRL_START[i++] = strlen(structDados->fileName) + 1; // strlen(structDados.fileName) + 1
bytes
        int j;
        for (j = 0; j < strlen(structDados->fileName) + 1; j++) {
                CTRL_START[i++] = structDados->fileName[j];
        }
        CTRL_START[i++] = 0x02; // A enviar comprimento máximo do campo de dados das tramas I
        CTRL_START[i++] = 0x02; // 2 bytes
        CTRL_START[i++] = (structDados->maxSize >> 8) & 0xff;
        CTRL_START[i++] = structDados->maxSize & 0xff;
```

Agora, algo semelhante é feito para a imagem propriamente dita. Dentro de um ciclo (que termina depois de toda a imagem ter sido lida), é criado um array de comprimento igual ao comprimento máximo selecionado no inicio pelo utilizador, a imagem é lida e este array é passado à camada inferior. E finalmente o mesmo é enviado para o receptor, o que indica o final da transmissão.

```
        do {
                unsigned char DATA_PACK[structDados->maxSize];
                DATA_PACK[0] = 0x00; // Indica dados //
                DATA_PACK[1] = numPack++ % 256; // Número de sequência
                resRead = fread(DATA_PACK + 4, sizeof(unsigned char), structDados->maxSize - 4,
structDados->fp);
                DATA_PACK[2] = (resRead >> 8) & 0xff;
                DATA_PACK[3] = resRead & 0xff;

                if (linkwrite(DATA_PACK,structDados, 4 + resRead, Ns, estado, tentativaEnvio,
podeEnviar) < 0) return -1;
                Ns = (Ns + 1) % 2;
        } while (resRead);

unsigned char CTRL_STOP[1];
        CTRL_STOP[0] = 0x02; // Indica paragem //
        if (linkwrite(CTRL_STOP,structDados, 1, Ns, estado, tentativaEnvio, podeEnviar) < 0)
return -1;
        Ns = (Ns + 1) % 2;
```

Do lado do receptor, dentro de um ciclo que termina após a recepção da trama que indica final da transmissão, o programa começa por receber da camada de ligação de dados o pacote a tratar. No caso de ser encontrado algum erro na recepção, a trama é rejeitada. Se isso não acontecer, caso se trate de uma trama de início de transmissão, o tamanho do ficheiro é guardado, para futura verificação de integridade e o seu nome é usado para criar o ficheiro a preencher.

```
                else if (dataPackage[0] == START) { //
                                unsigned int i = 1;
```

```
                unsigned int j;

                if (dataPackage[i++] == 0x00 && dataPackage[i++] == 0x02) {
                        structDados->filesize = dataPackage[i++] << 8;
                        structDados->filesize += dataPackage[i++];
                }

                if (dataPackage[i++] == 0x01) {
                        unsigned int size = dataPackage[i++];
                        for (j = 0; j < size; j++) {
                                structDados->fileName[j] = dataPackage[i++];
                                printf("%c", structDados->fileName[j]);
                        }
                        printf("\n");
                }

                structDados->fp = fopen(structDados->fileName, "wb");

                if (dataPackage[i++] == 0x02 && dataPackage[i++] == 0x02) {
                        structDados->maxSize = dataPackage[i++] << 8;
                        structDados->maxSize += dataPackage[i++];
                }

                Nr = (Nr + 1) % 2;
                RR[2] = (Nr == 0) ? 0x01 : 0x21;//
                RR[3] = RR[1] ^ RR[2];
                write(structDados->fd, RR, 5);
        }
```

Caso se trate de uma trama de dados, os dados são escritos no novo ficheiro criado.

```
        else if (dataPackage[0] == DATA) { //
                int posPackage = 0;
                if (duplicate) printf("Duplicate frame!\n");
                while (posPackage < packageSize && !duplicate) {
                        fwrite(dataPackage + 4 + posPackage, sizeof(unsigned char), 1,
structDados->fp);
                        posPackage++;
                        numread++;
                }
                if (!duplicate) Nr = (Nr + 1) % 2;
                duplicate = FALSE;
                RR[2] = (Nr == 0) ? 0x01 : 0x21;//
                RR[3] = RR[1] ^ RR[2];
                write(structDados->fd, RR, 5);
        }
```

Caso se trate de uma trama de final de transmissão, o ciclo interrompe-se e é então chamada a função fclose() para terminar a ligação entre emissor/receptor.

```
else if (dataPackage[0] == END) { //
                                    Nr = (Nr + 1) % 2;
                                    RR[2] = (Nr == 0) ? 0x01 : 0x21;//
                                    RR[3] = RR[1] ^ RR[2];
                                    write(structDados->fd, RR, 5);
                                    break;
                            }
                            } while (TRUE);
                    fclose(structDados->fp);
```

Validação

À medida que a implementação foi ganhando forma, foram feitos vários testes de robustez ao programa, como por exemplo: a capacidade de recepção e envio da imagem *pinguim.gif,* mediante condições normais e ainda caso se desligasse e voltasse novamente a ligar o cabo.

Em ambas as situações a resposta do programa foi a esperada, conseguindo corretamente superar tanto a existência de ficheiros duplicados, como a interrupção no envio dos pacotes. A imagem final do *pinguim.gif* foi assim obtida com sucesso.

Elementos de valorização

A interface implementada é extremamente flexível, permitindo ao utilizador escolher diferentes valores de baudrate, qual o nome e tamanho do ficheiro a transferir, a quantidade de vezes que o mesmo é transmitido e ainda qual o tamanho do timeout utilizado.

Foi ainda implementado um mecanismo de resposta REJ, na parte do receptor, no caso de falha na transferência dos pacotes de dados. Esta foi bastante simples pois a máquina de estado já estava implementada embora tivéssemos alterado para caso falhe num estado, isto é, caso não passe para o outro estado, retorna -1 à função linkread e usa o tcflush de modo a ignorar tudo o que estava na porta série à espera de ser lido. A função llread, ao receber, envia um REJ imediatamente ao emissor.

Conclusões

A transferência de dados através do uso de uma porta série verificou-se inicialmente um processo de difícil compreensão. No entanto, há medida que os problemas iam sendo resolvidos e que consolidamos o nosso programa, este tornou-se gratificante, visto que nos proporcionou um fantástico método de aprendizagem. Aprendemos ainda a estruturar corretamente um projeto em diferentes camadas/classes, permitindo assim padronizá-lo. Ao longo do relatório foram explicadas as principais funções chave do nosso programa bem como, o modo de utilização do mesmo.

Concluindo, foi assim possível, através da elaboração da aplicação, cimentar os nossos conhecimentos em programação em C e consolidar os conceitos dados nas aulas.

Anexo 1 - Código Fonte

app.c:

```
#include "app.h"
#include "establishment.h"
#include "data_transfer.h"
#include "termination.h"

#include <fcntl.h>
#include <termios.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define FALSE 0
#define TRUE 1

struct termios oldtio;
int duplicate = FALSE;


/*
 *  Open serial port device for reading
 */
int llopen(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar) {
	struct termios newtio;

	/* Open serial port device for reading and writing and not as controlling tty
	because we don't want to get killed if linenoise sends CTRL-C. */
	int fd = open(structDados->port, O_RDWR | O_NOCTTY);
	if (fd < 0) {
	perror(structDados->port); exit(-1); }

		if (tcgetattr(fd, &oldtio) == -1) { // save current port settings
			perror("tcgetattr");
			return -1;
		}

		bzero(&newtio, sizeof(newtio));
		newtio.c_cflag = structDados->baudRate | CS8 | CLOCAL | CREAD;
```

```
	newtio.c_iflag = IGNPAR;
	newtio.c_oflag = 0;

	/* set input mode (non-canonical, no echo,...) */
	newtio.c_lflag = 0;

	newtio.c_cc[VTIME] = 10; // inter-character timer
	newtio.c_cc[VMIN] = 0; // blocking read until x 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");
		return -1;
	}

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

	if (structDados->sender == TRUE)
		sendSET(fd, structDados, estado, tentativaEnvio, podeEnviar);
	else if (structDados->sender == FALSE)
		sendUA(fd, structDados, estado);

	return fd;
}


/*
 *
 */
int llwrite(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar) {
	int Ns = 0;
	unsigned int i = 0;

	unsigned char CTRL_START[11 + strlen(structDados->fileName) + 1];
	CTRL_START[i++] = 0x01; // Indica início
	CTRL_START[i++] = 0x00; // A enviar tamanho do ficheiro
	CTRL_START[i++] = 0x02; // 2 bytes
	CTRL_START[i++] = (unsigned char)((structDados->filesize & 0xff00) >> 8); // Tamanho do ficheiro
	CTRL_START[i++] = (unsigned char)(structDados->filesize & 0xff);
	CTRL_START[i++] = 0x01; // A enviar nome do ficheiro
	CTRL_START[i++] = strlen(structDados->fileName) + 1; // strlen(structDados.fileName) + 1 bytes
	int j;
	for (j = 0; j < strlen(structDados->fileName) + 1; j++) {
		CTRL_START[i++] = structDados->fileName[j];
```

```
        }
        CTRL_START[i++] = 0x02; // A enviar comprimento máximo do campo de dados das tramas I
        CTRL_START[i++] = 0x02; // 2 bytes
        CTRL_START[i++] = (structDados->maxSize >> 8) & 0xff;
        CTRL_START[i++] = structDados->maxSize & 0xff;

        if (linkwrite(CTRL_START,structDados, 11 + strlen(structDados->fileName) + 1, Ns, estado, tentativaEnvio,
podeEnviar) < 0) return -1;
        Ns = (Ns + 1) % 2;

        int resRead;
        int numPack = 0;
        int numWritten = 0;

        do {
                unsigned char DATA_PACK[structDados->maxSize];
                DATA_PACK[0] = 0x00; // Indica dados //
                DATA_PACK[1] = numPack++ % 256; // Número de sequência
                resRead = fread(DATA_PACK + 4, sizeof(unsigned char), structDados->maxSize - 4,
structDados->fp);
                DATA_PACK[2] = (resRead >> 8) & 0xff;
                DATA_PACK[3] = resRead & 0xff;

                if (linkwrite(DATA_PACK,structDados, 4 + resRead, Ns, estado, tentativaEnvio, podeEnviar) < 0)
return -1;
                Ns = (Ns + 1) % 2;
        } while (resRead);

        unsigned char CTRL_STOP[1];
        CTRL_STOP[0] = 0x02; // Indica paragem //
        if (linkwrite(CTRL_STOP,structDados, 1, Ns, estado, tentativaEnvio, podeEnviar) < 0) return -1;
        Ns = (Ns + 1) % 2;

        fclose(structDados->fp);

        return numWritten; // Número de bytes escritos
}


/*
 *
 */
int llread(Settings* structDados) {
        int Nr = 0;
        int numread = 0;
        unsigned char RR[5];
        RR[0] = FLAG;
        RR[1] = A;
```

```
RR[4] = FLAG;

unsigned char REJ[5];
REJ[0] = FLAG;
REJ[1] = A;
REJ[4] = FLAG;

do {
        unsigned char dataPackage[structDados->maxSize];

        int packageSize = linkread(dataPackage,structDados, duplicate);

        if (packageSize == -1) {
                REJ[2] = (Nr == 0) ? 0x05 : 0x25;
                REJ[3] = REJ[1] ^ REJ[2];
                tcflush(structDados->fd, TCIFLUSH);
                write(structDados->fd, REJ, 5);
        }
        else if (dataPackage[0] == DATA) { //
                int posPackage = 0;
                if (duplicate) printf("Duplicate frame!\n");
                while (posPackage < packageSize && !duplicate) {
                        fwrite(dataPackage + 4 + posPackage, sizeof(unsigned char), 1,
structDados->fp);
                        posPackage++;
                        numread++;
                }
                if (!duplicate) Nr = (Nr + 1) % 2;
                duplicate = FALSE;
                RR[2] = (Nr == 0) ? 0x01 : 0x21;//
                RR[3] = RR[1] ^ RR[2];
                write(structDados->fd, RR, 5);
        }
        else if (dataPackage[0] == START) { //
                unsigned int i = 1;
                unsigned int j;

                if (dataPackage[i++] == 0x00 && dataPackage[i++] == 0x02) {
                        structDados->filesize = dataPackage[i++] << 8;
                        structDados->filesize += dataPackage[i++];
                }

                if (dataPackage[i++] == 0x01) {
                        unsigned int size = dataPackage[i++];
                        for (j = 0; j < size; j++) {
                                structDados->fileName[j] = dataPackage[i++];
                                printf("%c", structDados->fileName[j]);
                        }
```

```
                        printf("\n");
                    }

                    structDados->fp = fopen(structDados->fileName, "wb");

                    if (dataPackage[i++] == 0x02 && dataPackage[i++] == 0x02) {
                        structDados->maxSize = dataPackage[i++] << 8;
                        structDados->maxSize += dataPackage[i++];
                    }

                    Nr = (Nr + 1) % 2;
                    RR[2] = (Nr == 0) ? 0x01 : 0x21;//
                    RR[3] = RR[1] ^ RR[2];
                    write(structDados->fd, RR, 5);
                }
                else if (dataPackage[0] == END) { //
                    Nr = (Nr + 1) % 2;
                    RR[2] = (Nr == 0) ? 0x01 : 0x21;//
                    RR[3] = RR[1] ^ RR[2];
                    write(structDados->fd, RR, 5);
                    break;
                }
            } while (TRUE);

            fclose(structDados->fp);

            FILE* filetest = fopen(structDados->fileName, "rb");
            fseek(filetest, 0, SEEK_END);
            if ((structDados->filesize = ftell(structDados->fp)))
                printf("\nFile sizes match!\n\n");
            else
                printf("\nFile sizes don't match!\n\n");

            return numread;
}

/*
 *
 */
int llclose(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar) {
            tentativaEnvio = 1;
            estado = 0;

            unsigned char DISC[5]; // Trama DISC (disconnect)
            DISC[0] = FLAG;
            DISC[1] = A;
            DISC[2] = 0x0B;
            DISC[3] = A ^ 0x0B;
```

```
        DISC[4] = FLAG;

        if (structDados->sender == TRUE)
                senderDISC(DISC, structDados, estado, tentativaEnvio, podeEnviar);
        else if (structDados->sender == FALSE)
                receiverDISC(DISC, structDados, estado);

        if (tcsetattr(structDados->fd, TCSANOW, &oldtio) == -1) {
                perror("tcsetattr");
                return -1;
        }

        printf("Closing port...\n");
        close(structDados->fd);

        return 0;
}
```

app.h

```c
#ifndef APP_H
#define APP_H


#include <stdio.h>

#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define FLAG 0x7e
#define A 0x03
#define DATA 0x00
#define START 0x01
#define END 0x02

/*
 * Struct that allows us saving all the information about the transmission
 */
typedef struct{


    int fd;
    FILE* fp;
    char* fileName;
    int filesize;
    int sender; // TRUE ou FALSE
    char port[20]; // Dispositivo /dev/ttySx
    int baudRate; // Velocidade de transmissão
    unsigned int timeout; // Valor do temporizador: 1 s
    unsigned int numTransmissions; // Número de tentativas em caso de falha
    unsigned int maxSize;

} Settings;


extern int llopen(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar);
extern int llwrite(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar);
extern int llread(Settings* structDados);
extern int llclose(Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int podeEnviar);

#endif
```

data_transfer.c

```
#include "data_transfer.h"

#include <fcntl.h>
#include <termios.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define FALSE 0
#define TRUE 1

unsigned char lastN = 255;


/*
 *
 */
int linkwrite(unsigned char* data, Settings* structDados, int datasize, int Ns, volatile int estado, volatile int tentativaEnvio,
volatile int podeEnviar) {
	tentativaEnvio = 1;
	podeEnviar = TRUE;
	estado = 0;

	unsigned int aSubstituir = 0;
	unsigned char FRAME_I[6 + datasize];
	FRAME_I[0] = FLAG;
	FRAME_I[1] = A;
	FRAME_I[2] = Ns == 0 ? 0x00 : 0x20;    //
	FRAME_I[3] = FRAME_I[1] ^ FRAME_I[2];
	unsigned char BCC2 = 0x00;
	int i;
	for (i = 0; i < datasize; i++) {
		FRAME_I[4 + i] = data[i];
		BCC2 ^= data[i];
		if (FRAME_I[4 + i] == 0x7e || FRAME_I[4 + i] == 0x7d) aSubstituir++;
	}
	FRAME_I[4 + datasize] = BCC2;
	if (FRAME_I[4 + datasize] == 0x7e || FRAME_I[4 + datasize] == 0x7d) aSubstituir++;
	FRAME_I[5 + datasize] = FLAG;
```

```
unsigned char FRAME_I_FINAL[6 + datasize + aSubstituir];
int j;
for (i = 0, j = 0; i < 6 + datasize + aSubstituir; i++, j++) {
        if (j < 4 || j == 6 + datasize - 1) FRAME_I_FINAL[i] = FRAME_I[j];
        else if (FRAME_I[j] == 0x7e || FRAME_I[j] == 0x7d) {
                FRAME_I_FINAL[i++] = 0x7d;
                FRAME_I_FINAL[i] = 0x20 ^ FRAME_I[j];
        }
        else FRAME_I_FINAL[i] = FRAME_I[j];
}

int OK = FALSE; // Recebeu RR corretamente?
while (tentativaEnvio <= structDados->numTransmissions) {
        if (OK) break;
        else if (podeEnviar) {
                printf("Sending frame: ");
                for (i = 0; i < 6 + datasize + aSubstituir; i++)
                        printf("%d ", FRAME_I_FINAL[i]);
                printf("\n");
                tcflush(structDados->fd, TCIFLUSH);
                write(structDados->fd, FRAME_I_FINAL, 6 + datasize + aSubstituir);
                alarm(structDados->timeout);
                podeEnviar = FALSE;
        }

        unsigned char c;
        int res = read(structDados->fd, &c, 1);

        if (res != 0) {
                alarm(0);

                switch (estado) {
                case 0:
                        if (c == FLAG) {
                                estado = 1;
                                printf("Confirmation: Switching to state 1\n");
                        }
                        else printf("Confirmation: Remaining on state 0\n");
                        break;
                case 1:
                        if (c == A) {
                                estado = 2;
                                printf("Confirmation: Switching to state 2\n");
                        }
                        else if (c == FLAG)
                                printf("Confirmation: Remaining on state 1\n");
                        else {
                                estado = 0;
```

```
                printf("Confirmation: Switching to state 0\n");
            }
            break;
        case 2:
            if (c == (Ns == 0 ? 0x21 : 0x01)) { //
                estado = 3;
                printf("Confirmation: positive. Switching to state 3\n");
            }
            else if (c == 0x05 || c == 0x25) { //
                podeEnviar = TRUE;
                estado = 0;
                printf("Confirmation: negative. Resending...\n");
            }
            else if (c == FLAG) {
                estado = 1;
                printf("Confirmation: Switching to state 1\n");
            }
            else {
                estado = 0;
                printf("Confirmation: Switching 2 state 0\n");
            }
            break;
        case 3:
            if (c == (A ^ (Ns == 0 ? 0x21 : 0x01))) { //
                estado = 4;
                printf("Confirmation: Switching to state 4\n");
            }
            else if (c == FLAG) {
                estado = 1;
                printf("Confirmation: Switching to state 1\n");
            }
            else {
                estado = 0;
                printf("Confirmation: Switching to state 0\n");
            }
            break;
        case 4:
            if (c == FLAG) {
                estado = 5;
                printf("Confirmation: Switching to state 5\n");
                OK = TRUE; // Sair do ciclo
            }
            else {
                estado = 0;
                printf("Confirmation: Switching to state 0\n");
            }
            break;
    }
```

```
                }
        }

        return tentativaEnvio <= structDados->numTransmissions ? 0 : -1;
}



/*
 *
 */
int linkread(unsigned char* dataPackage, Settings* structDados, int duplicate) {

        unsigned char c;
        unsigned char buf[structDados->maxSize + 6];
        int estado = 0;
        unsigned char bcc2 = 0x00;
        int packagesLength = 0;
        int i = 0;
        unsigned char Controlo;
        unsigned char N;

        int wasREJsent = FALSE;

        while (estado != 7 && wasREJsent == FALSE) {
                read(structDados->fd, &c, 1);

                switch (estado) {
                case 0:
                        if (c == FLAG)
                                buf[estado++] = c;
                        else
                                wasREJsent = TRUE;
                        break;
                case 1:
                        if (c == A)
                                buf[estado++] = c;
                        else
                                wasREJsent = TRUE;
                        break;
                case 2:
                        if (c == 0x00 || c == 0x20) //
                                buf[estado++] = c;
                        else
                                wasREJsent = TRUE;
                        break;
                case 3:
                        if (c == (buf[1] ^ buf[2]))
```

```
                buf[estado++] = c;
        else
                wasREJsent = TRUE;
        break;
case 4:
{
        int megaEstado = 0;
        if (c == DATA) {
                bcc2 ^= c;
                dataPackage[megaEstado++] = c; // C
                Controlo = c;

                read(structDados->fd, &c, 1);
                if (c == 0x7d) {
                        read(structDados->fd, &c, 1);
                        c ^= 0x20;
                }
                bcc2 ^= c;
                dataPackage[megaEstado++] = c; // N
                printf("N = %d\n", c);
                N = c + 1;

                read(structDados->fd, &c, 1);
                if (c == 0x7d) {
                        read(structDados->fd, &c, 1);
                        c ^= 0x20;
                }
                bcc2 ^= c;
                dataPackage[megaEstado++] = c; // L2
                read(structDados->fd, &c, 1);
                if (c == 0x7d) {
                        read(structDados->fd, &c, 1);
                        c ^= 0x20;
                }
                bcc2 ^= c;
                dataPackage[megaEstado++] = c; // L1
                packagesLength = 256 * dataPackage[2] + dataPackage[3];
                i = packagesLength;
                while (i > 0) {
                        read(structDados->fd, &c, 1);
                        if (c == 0x7d) {
                                read(structDados->fd, &c, 1);
                                c ^= 0x20;
                        }
                        bcc2 ^= c;
                        dataPackage[megaEstado++] = c;
                        i--;
                }
```

```
            estado = 5;
    }
    else if (c == START) {
            bcc2 ^= c;
            dataPackage[megaEstado++] = c;

            read(structDados->fd, &c, 1); // T1
            if (c == 0x7d) {
                    read(structDados->fd, &c, 1);
                    c ^= 0x20;
            }
            bcc2 ^= c;
            dataPackage[megaEstado++] = c;

            read(structDados->fd, &c, 1); // L1
            if (c == 0x7d) {
                    read(structDados->fd, &c, 1);
                    c ^= 0x20;
            }
            bcc2 ^= c;
            dataPackage[megaEstado] = c;

            for (i = dataPackage[megaEstado++]; i > 0; i--) {
                    read(structDados->fd, &c, 1); // V1
                    if (c == 0x7d) {
                            read(structDados->fd, &c, 1);
                            c ^= 0x20;
                    }
                    bcc2 ^= c;
                    dataPackage[megaEstado++] = c;
            }

            read(structDados->fd, &c, 1); // T2
            if (c == 0x7d) {
                    read(structDados->fd, &c, 1);
                    c ^= 0x20;
            }
            bcc2 ^= c;
            dataPackage[megaEstado++] = c;

            read(structDados->fd, &c, 1); // L2
            if (c == 0x7d) {
                    read(structDados->fd, &c, 1);
                    c ^= 0x20;
            }
            bcc2 ^= c;
            dataPackage[megaEstado] = c;
```

```
                for (i = dataPackage[megaEstado++]; i > 0; i--) {
                        read(structDados->fd, &c, 1); // V2
                        if (c == 0x7d) {
                                read(structDados->fd, &c, 1);
                                c ^= 0x20;
                        }
                        bcc2 ^= c;
                        dataPackage[megaEstado++] = c;
                }

                read(structDados->fd, &c, 1); // T3
                if (c == 0x7d) {
                        read(structDados->fd, &c, 1);
                        c ^= 0x20;
                }
                bcc2 ^= c;
                dataPackage[megaEstado++] = c;

                read(structDados->fd, &c, 1); // L3
                if (c == 0x7d) {
                        read(structDados->fd, &c, 1);
                        c ^= 0x20;
                }
                bcc2 ^= c;
                dataPackage[megaEstado] = c;

                for (i = dataPackage[megaEstado++]; i > 0; i--) {
                        read(structDados->fd, &c, 1); // V3
                        if (c == 0x7d) {
                                read(structDados->fd, &c, 1);
                                c ^= 0x20;
                        }
                        bcc2 ^= c;
                        dataPackage[megaEstado++] = c;
                }

                estado = 5;
        }
        else if (c == END) {
                bcc2 ^= c;
                dataPackage[megaEstado++] = c;
                estado = 5;
        }
        else
                wasREJsent = TRUE;
        break;
}
case 5:
```

```
                printf("bcc2 esperado: %d, obtido: %d\n", bcc2, c);
                if (c == 0x7d) {
                        read(structDados->fd, &c, 1);
                        c ^= 0x20;
                        if (c == bcc2)
                                estado = 6;
                        else
                                wasREJsent = TRUE;
                }
                else if (c == bcc2)
                        estado = 6;
                else
                        wasREJsent = TRUE;
                break;
        case 6:
                if (c == FLAG)
                        estado = 7;
                else
                        wasREJsent = TRUE;
                break;
        }
    }

    printf("WasRejSent: %d (bcc2 = %d)\n", wasREJsent, bcc2);
    if (wasREJsent) return -1;

    if (Controlo == 0x00) { // Se foi pacote de dados...
        if (N == lastN) duplicate = TRUE;
        lastN = N;
    }

    return packagesLength;
}
```

data_transfer.h

```
#ifndef DATA_TRANSFER_H
#define DATA_TRANSFER_H

#include "app.h"

#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define FLAG 0x7e
#define A 0x03
#define DATA 0x00
#define START 0x01
#define END 0x02

extern int linkwrite(unsigned char* data, Settings* structDados, int datasize, int Ns, volatile int estado, volatile int
tentativaEnvio, volatile int podeEnviar);
extern int linkread(unsigned char* dataPackage, Settings* structDados, int duplicate);

#endif
```

establishment.c

```c
#include "establishment.h"

#include <fcntl.h>
#include <termios.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define FALSE 0
#define TRUE 1

/*
 * Function that send the SET-TRAMA
 */
void sendSET(int fd, Settings* structDados, int estado, int tentativaEnvio, int podeEnviar) {
	unsigned char SET[5]; // Trama SET
	SET[0] = FLAG;
	SET[1] = A;
	SET[2] = 0x07; //
	SET[3] = SET[1] ^ SET[2];
	SET[4] = FLAG;

	unsigned char c;
	unsigned char buf[5];
	(void) buf;
	int passed = FALSE;

	while (tentativaEnvio <= structDados->numTransmissions && passed == FALSE) {
		if (podeEnviar) {
			podeEnviar = FALSE;
			alarm(structDados->timeout);
			int res = write(fd, SET, 5);

			if (res == 5) printf("Frame sent successfully\n");
			else {
				printf("Frame wasn't sent successfully\n");
				podeEnviar = TRUE;
			}

			alarm(structDados->timeout);
```

```
while (estado != 5) {
        if (podeEnviar) {
                alarm(0);
                break;
        }

        read(fd, &c, 1);

        switch (estado) {
        case 0:
                if (c == FLAG) {
                        estado = 1;
                        printf("UA: Switching to state 1\n");
                        buf[0] = c;
                }
                else printf("UA: Remaining on state 0\n");
                break;
        case 1:
                if (c == A) {
                        estado = 2;
                        printf("UA: Switching to state 2\n");
                        buf[1] = c;
                }
                else if (c == FLAG) {
                        estado = 1;
                        printf("UA: Remaining on state 1\n");
                        buf[0] = c;
                }
                else {
                        estado = 0;
                        printf("UA: Switching to state 0\n");
                }
                break;
        case 2:
                if (c == 0x03) { //
                        estado = 3;
                        printf("UA: Switching to state 3\n");
                        buf[2] = c;
                }
                else if (c == FLAG) {
                        estado = 1;
                        printf("UA: Switching to state 1\n");
                        buf[0] = c;
                }
                else {
                        estado = 0;
                        printf("UA: Switching to state 0\n");
```

```
                                        }
                                        break;
                                case 3:
                                        if (c == (A ^ 0x03)) {  //
                                                estado = 4;
                                                printf("UA: Switching to state 4\n");
                                                buf[3] = c;
                                        }
                                        else if (c == FLAG) {
                                                estado = 1;
                                                printf("UA: Switching to state 1\n");
                                                buf[0] = c;
                                        }
                                        else {
                                                estado = 0;
                                                printf("UA: Switching to state 0\n");
                                        }
                                        break;
                                case 4:
                                        if (c == FLAG) {
                                                alarm(0);
                                                estado = 5;
                                                printf("UA: Switching to state 5\n");
                                                buf[4] = c;
                                                passed = TRUE;
                                        }
                                        else {
                                                estado = 0;
                                                printf("UA: Switching to state 0\n");
                                        }
                                        break;
                                }
                        }

                }
        }
        if (tentativaEnvio > structDados->numTransmissions) exit(-1);
}


/*
 * Function that send the UA response
 */
void sendUA(int fd, Settings* structDados, int estado) {
        unsigned char UA[5]; // Trama UA (unnumbered acknowledgement)
        UA[0] = FLAG;
        UA[1] = A;
        UA[2] = 0x03; //
```

```
UA[3] = A ^ 0x03; //
UA[4] = FLAG;

unsigned char c;
unsigned char buf[5];
(void) buf;

while (estado != 5) {
        read(fd, &c, 1);

        switch (estado) {
        case 0:
                if (c == FLAG) {
                        estado = 1;
                        printf("SET: Switching to state 1\n");
                        buf[0] = c;
                }
                else printf("SET: Remaining on state 0\n");
                break;
        case 1:
                if (c == A) {
                        estado = 2;
                        printf("SET: Switching to state 2\n");
                        buf[1] = c;
                }
                else if (c == FLAG) {
                        estado = 1;
                        printf("SET: Remaining on state 1\n");
                        buf[0] = c;
                }
                else {
                        estado = 0;
                        printf("SET: Switching to state 0\n");
                }
                break;
        case 2:
                if (c == 0x07) { //
                        estado = 3;
                        printf("SET: Switching to state 3\n");
                        buf[2] = c;
                }
                else if (c == FLAG) {
                        estado = 1;
                        printf("SET: Switching to state 1\n");
                        buf[0] = c;
                }
                else {
                        estado = 0;
```

```
                        printf("SET: Switching to state 0\n");
                    }
                    break;
                case 3:
                    if (c == (A ^ 0x07)) { //
                        estado = 4;
                        printf("SET: Switching to state 4\n");
                        buf[3] = c;
                    }
                    else if (c == FLAG) {
                        estado = 1;
                        printf("SET: Switching to state 1\n");
                        buf[0] = c;
                    }
                    else {
                        estado = 0;
                        printf("SET: Switching to state 0\n");
                    }
                    break;
                case 4:
                    if (c == FLAG) {
                        estado = 5;
                        printf("SET: Switching to state 5\n");
                        buf[4] = c;
                    }
                    else {
                        estado = 0;
                        printf("SET: Switching to state 0\n");
                    }
                    break;
                }
            }

            write(fd, UA, 5);
}
```

establishment.h

```
#ifndef ESTABLISHMENT_H
#define ESTABLISHMENT_H

#include "app.h"

#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define FLAG 0x7e
#define A 0x03

extern void sendSET(int fd, Settings* structDados, int estado, int tentativaEnvio, int podeEnviar);
extern void sendUA(int fd, Settings* structDados, int estado);

#endif
```

main.c

```
#include "app.h"

#include <fcntl.h>
#include <termios.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define FALSE 0
#define TRUE 1

static Settings* structDados; // Esta definida no app.h

volatile int estado = 0, tentativaEnvio = 1, podeEnviar = TRUE;

/*
 *
 */
void atendeAlarme()  {
        printf("Bad response #%d from receiver!\n", tentativaEnvio);
        tentativaEnvio++;
        podeEnviar = TRUE;
        estado = 0;
}

/*
 *
 */
void limparEcra() {
        unsigned int i;
        for (i = 0; i < 50; i++)
                printf("\n");
};



/*
 * Prepare the port to send the packages
 */
void prepareSender() {
        limparEcra();
```

```
	if ((structDados->fp = fopen(structDados->fileName, "rb")) != NULL) {
		fseek(structDados->fp, 0, SEEK_END);
		structDados->filesize = ftell(structDados->fp);
		rewind(structDados->fp);
	}
	else {
		printf("File not found!\n");
		exit(-1);
	}

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

	llwrite(structDados, estado, tentativaEnvio, podeEnviar);
}

/*
 * Prepare the port to receive the packages
 */
void prepareReceiver() {
	limparEcra();

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

	//unsigned char* buf = NULL;
	llread(structDados);
}

/*
 * Get the information about the transmission
 */
void startstruct() {

	int choicePort = -1;


	structDados = malloc(sizeof(Settings));
	structDados->fp = malloc(sizeof(FILE));
	structDados->fd =  0;
	structDados->fileName = malloc(255 * sizeof(char) + 1);
	structDados->filesize = 0;
	structDados->sender = 0;
	/* structDados->port = malloc(20 * sizeof(char) + 1); doesnt need */
	structDados->baudRate = 0;
	structDados->timeout = 0;
	structDados->numTransmissions = 0;
	structDados->maxSize = 0;
```

```
do {
        limparEcra();
        printf(". Are you a sender or a receiver?\n");
        printf("    1. Sender \n");
        printf("    2. Receiver \n");
        char choice = getchar();

        if (choice == '1'){
                structDados->sender = TRUE;
                break;
        }
        else if (choice == '2'){
                structDados->sender = FALSE;
                break;
        }
        else
                continue;

} while (TRUE);


if (structDados->sender == TRUE) {
        limparEcra();
        printf(". What's the name of the file you want to send?\n");
        getchar();
        gets(structDados->fileName);
}



limparEcra();
printf(". Which port will you use?\n");
scanf("%d", &choicePort);
snprintf(structDados->port, sizeof(structDados->port), "/dev/ttyS%d", choicePort);
sleep(1);

do {
        limparEcra();
        printf(". Which BAUDRATE do you prefer?\n");
        printf("     1. B9600\n");
        printf("     2. B19200\n");
        printf("     3. B38400\n");
        printf("     4. B57600\n");
        printf("     5. B115200\n");
        int choice = scanf("%d", &choice);
        if (choice == 1) {
                structDados->baudRate = B9600;
                break;
```

```
                }
                else if (choice == 2) {
                        structDados->baudRate = B19200;
                        break;
                }
                else if (choice == 3) {
                        structDados->baudRate = B38400;
                        break;
                }
                else if (choice == 4) {
                        structDados->baudRate = B57600;
                        break;
                }
                else if (choice == 5) {
                        structDados->baudRate = B115200;
                        break;
                }
                else continue;
        } while (TRUE);

        limparEcra();
        printf(". How will your timeout be?\n");
        scanf("%d", &structDados->timeout);

        limparEcra();
        printf(". How many times will your program try to send a package?\n");
        scanf("%d", &structDados->numTransmissions);

        if (structDados->sender) {
                limparEcra();
                printf(". How many data bytes should each frame contain?\n");
                scanf("%d", &structDados->maxSize);
                structDados->maxSize += 4;
        }
        else structDados->maxSize = 255;
}


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

        signal(SIGALRM, atendeAlarme); /* Instalar alarme */
        setvbuf(stdout, NULL, _IONBF, 0); /* Desativar buffer do STDOUT */

        do {

                limparEcra();

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

```
            printf("- SERIAL PORT RCOM 1516 -\n");
            printf("-------------------------\n");
            printf("- 1. Run                                -\n");
            printf("- 2. Exit              -\n");
            printf("-------------------------\n");

            int choice;
            scanf("%d", &choice);

            switch (choice) {
            case 1:
                  startstruct();
                  structDados->fd = llopen(structDados, estado, tentativaEnvio, podeEnviar);

                  if (structDados->sender)
                        prepareSender();
                  else
                        prepareReceiver();

                  llclose(structDados, estado, tentativaEnvio, podeEnviar);
                  return 0;
            case 2:
                  printf("Exiting program! \n");
                  return 0;
            default:
                  continue;
            }

      } while (TRUE);

      return 0;
}
```

termination.c

```c
#include "termination.h"

#include <fcntl.h>
#include <termios.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

#define FALSE 0
#define TRUE 1


/*
 *
 */
void senderDISC(unsigned char* DISC, Settings* structDados, volatile int estado, volatile int tentativaEnvio, volatile int
podeEnviar) {
	tentativaEnvio = 1;
	podeEnviar = TRUE;
	estado = 0;

	unsigned char c;
	unsigned char buf[5];
	(void) buf;

	while (tentativaEnvio <= structDados->numTransmissions) {
		if (podeEnviar) {
			podeEnviar = FALSE;
			alarm(structDados->timeout);
			int res = write(structDados->fd, DISC, 5);

			if (res == 5) printf("Sent DISC\n");
			else {
				printf("Unable to send DISC\n");
				podeEnviar = TRUE;
			}

			alarm(structDados->timeout);

			while (estado != 5) {
```

```
if (podeEnviar) {
		alarm(0);
		break;
}

read(structDados->fd, &c, 1);

switch (estado) {
case 0:
		if (c == FLAG) {
			estado = 1;
			printf("DISC: Switching to state 1\n");
			buf[0] = c;
		}
		else printf("DISC: Remaining on state 0\n");
		break;
case 1:
		if (c == A) {
			estado = 2;
			printf("DISC: Switching to state 2\n");
			buf[1] = c;
		}
		else if (c == FLAG) {
			estado = 1;
			printf("DISC: Remaining on state 1\n");
			buf[0] = c;
		}
		else {
			estado = 0;
			printf("DISC: Switching to state 0\n");
		}
		break;
case 2:
		if (c == 0x0B) {
			estado = 3;
			printf("DISC: Switching to state 3\n");
			buf[2] = c;
		}
		else if (c == FLAG) {
			estado = 1;
			printf("DISC: Switching to state 1\n");
			buf[0] = c;
		}
		else {
			estado = 0;
			printf("DISC: Switching to state 0\n");
		}
		break;
```

```
                                        case 3:
                                                if (c == (A ^ 0x0B)) {
                                                        estado = 4;
                                                        printf("DISC: Switching to state 4\n");
                                                        buf[3] = c;
                                                }
                                                else if (c == FLAG) {
                                                        estado = 1;
                                                        printf("DISC: Switching to state 1\n");
                                                        buf[0] = c;
                                                }
                                                else {
                                                        estado = 0;
                                                        printf("DISC: Switching to state 0\n");
                                                }
                                                break;
                                        case 4:
                                                if (c == FLAG) {
                                                        alarm(0);
                                                        estado = 5;
                                                        printf("DISC: Switching to state 5\n");
                                                        buf[4] = c;
                                                        tentativaEnvio = structDados->numTransmissions + 1; // Obrigar a
sair do ciclo
                                                }
                                                else {
                                                        estado = 0;
                                                        printf("DISC: Switching to state 0\n");
                                                }
                                                break;
                                        }
                                }
                        }
                }

                unsigned char UA[5]; // Trama UA (unnumbered acknowledgement)
                UA[0] = FLAG;
                UA[1] = A;
                UA[2] = 0x03; //
                UA[3] = A ^ 0x03; //
                UA[4] = FLAG;

                write(structDados->fd, UA, 5);
                sleep(1);
}


/*
```

```
 *
 */
void receiverDISC(unsigned char* DISC, Settings* structDados, volatile int estado) {
	unsigned char c;
	unsigned char buf[5];
	(void) buf;

	while (estado != 5) {
		read(structDados->fd, &c, 1);

		switch (estado) {
		case 0:
			if (c == FLAG) {
				estado = 1;
				printf("DISC: Switching to state 1\n");
				buf[0] = c;
			}
			else printf("DISC: Remaining on state 0\n");
			break;
		case 1:
			if (c == A) {

				estado = 2;
				printf("DISC: Switching to state 2\n");
				buf[1] = c;
			}
			else if (c == FLAG) {
				estado = 1;
				printf("DISC: Remaining on state 1\n");
				buf[0] = c;
			}
			else {
				estado = 0;
				printf("DISC: Switching to state 0\n");
			}
			break;
		case 2:
			if (c == 0x0B) {
				estado = 3;
				printf("DISC: Switching to state 3\n");
				buf[2] = c;
			}
			else if (c == FLAG) {
				estado = 1;
				printf("DISC: Switching to state 1\n");
				buf[0] = c;
			}
			else {
```

```
                        estado = 0;
                        printf("DISC: Switching to state 0\n");
                }
                break;
        case 3:
                if (c == (A ^ 0x0B)) {
                        estado = 4;
                        printf("DISC: Switching to state 4\n");
                        buf[3] = c;
                }
                else if (c == FLAG) {
                        estado = 1;
                        printf("  DISC: Switching to state 1\n");
                        buf[0] = c;
                }
                else {
                        estado = 0;
                        printf("DISC: Switching to state 0\n");
                }
                break;
        case 4:
                if (c == FLAG) {
                        estado = 5;
                        printf("DISC: Switching to state 5\n");
                        buf[4] = c;
                }
                else {
                        estado = 0;
                        printf("DISC: Switching to state 0\n");
                }
                break;
        }
}

write(structDados->fd, DISC, 5);

estado = 0;

while (estado != 5) {
        read(structDados->fd, &c, 1);

        switch (estado) {
        case 0:
                if (c == FLAG) {
                        estado = 1;
                        printf("UA: Switching to state 1\n");
                        buf[0] = c;
                }
```

```
            else printf("UA: Remaining on state 0\n");
            break;
case 1:
            if (c == A) {

                        estado = 2;
                        printf("UA: Switching to state 2\n");
                        buf[1] = c;
            }
            else if (c == FLAG) {
                        estado = 1;
                        printf("UA: Remaining on state 1\n");
                        buf[0] = c;
            }
            else {
                        estado = 0;
                        printf("UA: Switching to state 0\n");
            }
            break;
case 2:
            if (c == 0x03) { //
                        estado = 3;
                        printf("UA: Switching to state 3\n");
                        buf[2] = c;
            }
            else if (c == FLAG) {
                        estado = 1;
                        printf("UA: Switching to state 1\n");
                        buf[0] = c;
            }
            else {
                        estado = 0;
                        printf("UA: Switching to state 0\n");
            }
            break;
case 3:
            if (c == (A ^ 0x03)) { //
                        estado = 4;
                        printf("UA: Switching to state 4\n");
                        buf[3] = c;
            }
            else if (c == FLAG) {
                        estado = 1;
                        printf("UA: Switching to state 1\n");
                        buf[0] = c;
            }
            else {
                        estado = 0;
```

```
                        printf("UA: Switching to state 0\n");
                    }
                    break;
                case 4:
                    if (c == FLAG) {
                        estado = 5;
                        printf("UA: Switching to state 5\n");
                        buf[4] = c;
                    }
                    else {
                        estado = 0;
                        printf("UA: Switching to state 0\n");
                    }
                    break;
                }
            }
}
```

termination.h

```
#ifndef TERMINATION_H
#define TERMINATION_H

#include "app.h"

#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define FLAG 0x7e
#define A 0x03

extern void senderDISC(unsigned char* DISC, Settings* structDados, volatile int estado, volatile int tentativaEnvio,
volatile int podeEnviar);
extern void receiverDISC(unsigned char* DISC, Settings* structDados, volatile int estado);


#endif
```