Protocolo de ligação de dados

(1º Trabalho Laboratorial)

- Miguel Filipe Brandão Martins Lima up202108659

- Pedro Miguel Martins Romão up202108660

Sumário

O objetivo deste projeto é implementar um protocolo de nível de ligação de dados. Para alcançar esse objetivo, será desenvolvida uma aplicação simples de transferência de arquivos que permite acessar arquivos no disco (no lado do transmissor) e armazenar arquivos no disco (no lado do receptor) através da porta série RS-232.

Através deste projeto, conseguimos aplicar o conhecimento adquirido nas aulas teóricas para colocar em prática o protocolo em questão, consolidando conhecimentos como o funcionamento e a eficiência da estratégia Stop-and-Wait.

1. Introdução

O objetivo do trabalho foi desenvolver um protocolo de ligação de dados, de acordo com as especificações do guião, capaz de transferir um ficheiro de um computador para outro através da porta série. O relatório está divido em secções:

- **Arquitetura** - blocos funcionais e interfaces.
- **Estrutura do código** - principais estruturas de dados, funções e relação com a arquitetura.
- **Casos de uso principais** - lógica do codigo, sequências de chamada de funções.
- **Protocolo de ligação lógica** - Funcionamento da camada de ligação lógica e a sua implementação.
- **Protocolo de aplicação** - Funcionamento da camada de aplicação e a sua implementação.
- **Validação** - Testes efetuados para avaliar a robustez e o correcto funcionamento do protocolo.
- **Conclusões** - síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

2. Arquitetura

Blocos funcionais

O projeto foi desenvolvido tendo em conta dois principais blocos lógicos, `Linklayer` e a `Applicationlayer`.

A camada `LinkLayer` é responsável pelo funcionamento do protocolo da ligação de dados. É responsavel pelo estabelecimento e término da ligação, pela criação de tramas com informação e o seu envio através da porta série. É também responsável de verificação dos dados enviados e recebidos garantindo que os dados sejam os pretendidos.

A camada `ApplicationLayer` utiliza a API da LinkLayer para transferência e receção de pacotes de dados de um ficheiro. É a camada mais proxima do utilizador e nela pode se redifinir o tamanho das tramas de informação, o número de retransmições, e o tempo entre cada retransmição.

Interface

Para executar o programa é necessário dois terminais, um em cada computador. Um deles corre em modo emissor e o outro em modo recetor.

<'nome do binário'> <'porta'> <'modo'>

- ./program /dev/ttyS10 emissor
- ./program /dev/ttyS11 recetor

3. Estrutura do código

As funções e estruturas de dados utilizadas na `LinkLayer`:

```
// cria ligação com a serial port
int setup(const char *serialPortName);
// estabelece ligação com entre o emissor e recetor
int llopen(const char *porta, enum Status status);
// termina a ligação
int llclose(int fd);
// envia tramas
int llwrite(int fd, unsigned char *buffer, int length);
// lê as tramas
int llread(int fd, unsigned char *buffer);
```

```
 //estados das maquinas de estados
enum rec_status
{
    Start,
    flag_rcv,
    a_rcv,
    c_rcv,
    bcc_ok,
    a_tx,
    c_tx,
    STOP,
    read_data,
    found_esc,
    next_esc
};

//modo
enum Status
{
    RECEIVER,
    TRANSMITTER
};
```

As funções utilizadas na `ApplicationLayer`:

```
 //cria um pacote de controlo
unsigned char *MakeCPacket(unsigned char control, unsigned char *filename, long int length, unsigned int *size);
//cria um pacote de dados
unsigned char *MakeDPacket(unsigned char control, unsigned char *data, int length, unsigned int *size);
```

A lógica da AplicationLayer está na função main da application_layer.c pelo que estas funções são apenas auxiliares. O código implementado para o devido funcionamento da application_layer está dentro da função main.

4. Casos de uso principais

Sendo o principal objetivo do trabalho a criação de uma aplicação que transfira ficheiros entre duas máquinas através da porta série os utilizadores teram que escolher certos parâmetros para poder correr o programa.

Execução do programa

Do lado do emissor terá que escolher a porta série em uso. Deste modo o programa poderá ser executado do seguinte modo:

- ./program <porta série> emissor
- ./program /dev/tty/S10 emissor

Do lado do receptor apenas terá que escolher também o número que representa a porta série em uso. Deste modo o programa poderá ser executado do seguinte modo:

- ./program <porta série> receptor
- (exemplo)./program /dev/tty/S11 receptor

Programa executado como emissor

Uma vez inseridos os argumentos na linha de comandos o programa executará código simples para deteção do utilizador como emissor, desencadeando a seguinte sequência de chamadas: • Estabelecimeto da ligação com auxílio de llopen. • Envio do pacote de controlo de start com llwrite. • Leitura do ficheiro para um char pointer. • Envios sucessivos dos pacotes de dados correspondentes ao conteúdo lido com llwrite. • Envio do pacote de controlo END com sendControlPacket • Encerramento da ligação com a chamada de llclose

Programa executado como receptor

Uma vez inseridos os argumentos na linha de comandos o programa executará a mesma distinção de utilizador, desencadeando a seguinte sequência de chamadas: • Estabelecimeto da ligação com auxílio de llopen. • Chamadas sucessivas de llread. • Por cada chamada de llread informação lida é escrita no fichero novo. • Encerramento da ligação com a chamada de llclose.

5 Protocolo de ligação lógica

5.1 llopen

`int llopen ( char * port , conn_type connection_type ) ;` Função responsável por estabelecer uma ligação através da porta série. Quando a função é

invocada pelo emissor é enviado o comando SET através da porta série e é aguardada uma resposta por parte do recetor, comando UA. Caso o tempo de expera pela resposta exceda o time out definido, o comando SET é reenviado ficando o emissor novamente à espera de nova resposta. Este mecanismo é conseguido graças à implementação de um alarme e caso o número de tentativas de envio do comando SET excedam um limite definido o emissor encerra. Caso a função seja invocada pelo receptor, este espera pela receção do comando SET e de seguida envia o comando UA.

5.2 llwrite

`int llwrite ( int fd , char * buffer , int length ) ;` Função invocada pelo emissor e é responsável pelo envio de tramas de infor- mação, receção de respostas de reconhecimento por parte do receptor e garante o cumprimento do protocolo de ligação de dados. Os packets para enviar são construídos pelas funções makeXPacket onde X pode ser D ou C , para pacote de informação ou de controlo. A função llwrite é também responsável pelo stuffing da trama. O envio da trama de informação possui o mesmo mecanismo de time out referido na função llopen. Neste caso o alarme é acionado quando a mensagem é enviada até à receção de uma resposta RR ou REJ. Caso o comando recebido seja do tipo REJ, a trama de informação anteriormente enviada é reenviada, garantindo que não há perdas de informação.

5.3 llread

`int llread ( int fd , char * buffer ) ;` Função invocada pelo receptor e é responsável por receber e interpretar as tramas de infor- mação através da porta série, enviando uma resposta adequada ao emissor. É realizado destuffing da mensagem recebida. Uma vez que tramas com erros no cabeçalho são automaticamente descartadas, devido à implementação da máquina de estados, a função apenas verifica se o valor do BCC2 se encontra correto, enviando REJ caso não se verifique. O campo de dados da trama de informação é retornado através do paraâmetro char* buffer para futura interpretação por parte da camada da aplicação.

5.4 llclose

`int llclose ( int fd ) ;` Função responsável por encerrar a ligação através da porta série, tendo um comportamento semelhante à função llopen com a mesma implementação de alarme, diferindo nos comandos enviados e no facto de apenas ser chamada pelo emissor. Quando a função é invocada pelo emissor é enviado o comando DISC através da porta série através da função sendcontrol e é aguardada uma resposta por parte do recetor, comando DISC. Finalmente é enviado o comando UA. O receptor ao receber DISC na seqência de llreads de seguida envia o comando DISC. Finalmente ao receber o comando UA, a ligação através da porta série é efe- tivamente terminada.

6. Protocolo de aplicação

A camada de aplicação desempenha um papel crucial na interação direta com o ficheiro a ser transferido e o utilizador. Nessa camada, é possível definir várias configurações, como o arquivo a ser transferido, em que porta serie, a velocidade da transferência (BAUDRATE), o número de bytes de dados inseridos em cada pacote, o número máximo de retransmissões e o tempo máximo de espera pela resposta do receptor. A transferência do arquivo ocorre por meio da API da camada de enlace de dados (LinkLayer), que traduz os dados em pacotes de informação.

Após a realização do handshake entre o transmissor e o receptor, o conteúdo completo do ficheiro é copiado para um buffer local e, em seguida, fragmentado pela camada de aplicação com base no número de bytes especificado. O primeiro pacote enviado pelo transmissor contém dados formatados em formato TLV (Type, Length, Value), criados pela função "MakeCPacket", onde são passados como argumento o tamanho do ficheiro em bytes e o nome do ficheiro. No lado do receptor, esse pacote é descompactado para criar e alocar o espaço necessário para receber o ficheiro.

Cada fragmento do ficheiro a ser transferido é inserido em um pacote de dados e enviado pela porta serie utilizando a função "llwrite" presente na API. Cada envio é seguido por uma resposta por parte do receptor, indicando se o pacote enviado foi aceite ou rejeitado. No primeiro caso, o transmissor envia o próximo fragmento, e no segundo caso, ele reenvia o mesmo fragmento. Cada pacote é avaliado individualmente pelo receptor por meio da função "llread". Se a informação presente no pacote de dados for recebida corretamente no recetor, essa informação é escrita em um ficheiro novo criado.

A conexão entre as duas máquinas é encerrada quando a função "llclose" da API é invocada, seja após a conclusão da transferência de pacotes de dados ou se o número de tentativas for excedido.

7. Validação

Para garantir a implementação correta do protocolo desenvolvido, realizamos diversos testes ao longo do projeto. Estes testes incluíram:

- Transferência de arquivos de diferentes tamanhos.
- Transferência de arquivos com nomes distintos.
- Transferência de arquivos com diferentes taxas de transmissão (baud rates).
- Transferência de pacotes de dados de tamanhos variados.
- Introdução de interrupções totais na porta serial.
- Introdução de ruído na porta serial por meio de curto-circuitos (Feito pelo docente).

Em todos esses cenários, o protocolo "Stop-And-Wait" que implementamos garantiu a consistência dos arquivos transferidos. Além disso, durante a apresentação do projeto na aula laboratorial, os testes foram reproduzidos na presença do docente, o que ajudou a validar ainda mais a eficácia e confiabilidade da implementação do protocolo. Isso demonstra a robustez e a capacidade do protocolo de lidar com diferentes condições adversas e garantir a integridade dos dados transferidos.

8.Eficiência do protocolo de ligação de dados

Caracterização teórica stop and wait

Do ponto de vista teórico, o protocolo Stop Wait consiste no emissor aguardar um reconhecimento positivo (ACK) do receptor após a transmissão de uma trama de dados. Ao receber o ACK, o emissor sabe que a trama foi transmitida com sucesso e pode prosseguir para enviar outra. Em caso de erro, o receptor envia um reconhecimento negativo (NACK) para instruir o emissor a reenviar a trama.

O nosso programa baseia-se fortemente neste mecanismo de Stop Wait. Quando o emissor envia uma trama, ele espera uma resposta. Se o receptor receber os dados sem erros, envia uma mensagem RR positiva, sinalizando ao emissor para enviar outra trama. Por outro lado, uma mensagem REJ negativa indica a necessidade de reenvio.

O campo Nr nestas tramas de resposta varia dependendo de se o emissor enviou uma trama Ns de 0 ou 1. Isso ajuda o emissor a determinar qual trama enviar e auxilia no tratamento de tramas repetidas. Com Ns=0, o emissor espera ou um RR (Nr=1) para recepção sem erros ou um REJ (Nr=0) para erros. Para Ns=1, o

emissor espera um RR (Nr=0) para recepção sem erros ou um REJ (Nr=1) para erros.

9. Conclusões

O protocolo de ligação de dados `LinkLayer`, responsável pela interação com a porta serie e pela gestão das tramas de informação, e o protocolo de aplicação `ApplicationLayer`, que interage diretamente com o arquivo a ser transferido, desempenhou um papel fundamental na consolidação dos conceitos abordados nas aulas teóricas. Através deste projeto, conseguimos assimilar e aplicar os conceitos de byte stuffing, criação de frames e o funcionamento do protocolo Stop-and-Wait. Além disso, aprendemos como esse protocolo é capaz de detectar erros e lidar com eles de maneira eficaz. Isso contribuiu significativamente para nossa compreensão prática dos princípios de comunicação de dados e para a consolidação do conhecimento adquirido nas aulas teóricas.

Anexo 1

link_layer.h

```
#ifndef _LINK_LAYER_H_
#define _LINK_LAYER_H_

#include <stdio.h>


// Define your constants here
#define BAUDRATE B38400
#define _POSIX_SOURCE 1 // POSIX compliant source

#define FALSE 0
#define TRUE 1

#define BUF_SIZE 5 // 256

#define FLAG 0x7E
#define A_RECEIVER 0x01
#define A_TRANSMITTER 0x03
#define SET 0x03
#define UA 0x07
#define RR0 0x05
#define RR1 0x85
#define REJ0 0x01
#define REJ1 0x81
#define DISC 0x0B
#define INFORMATION_FRAME0 0x00
#define INFORMATION_FRAME1 0x0B
//#define MAX_PAYLOAD_SIZE 40




#define C_I(Ns) (Ns << 6)

#define C_RR(Nr) ((Nr << 7) | 0x05)
#define C_REJ(Nr) ((Nr << 7) | 0x01)

#define ESC 0x7d
#define FLAG_R 0x5e
#define ESC_R 0x5d

#define C_DISC 0x0B


enum rec_status
{
    Start,
    flag_rcv,
    a_rcv,
    c_rcv,
    bcc_ok
```

```
    bcc_ok,
    a_tx,
    c_tx,
    STOP,
    read_data,
    found_esc,
    next_esc
};



enum Status
{
    RECEIVER,
    TRANSMITTER
};




// cria ligação com a serial port
int setup(const char *serialPortName);
// estabelece ligação com entre o emissor e recetor
int llopen(const char *porta, enum Status status);
// termina a ligação
int llclose(int fd);
// envia tramas
int llwrite(int fd, unsigned char *buffer, int length);
// lê as tramas
int llread(int fd, unsigned char *buffer);

#endif // LINK_LAYER_H
```

link_layer.c

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

#include <signal.h>
#include <stdio.h>

#include <stdbool.h>

#include "link_layer.h"



int delay = 3;
int numretransmitions = 3;

int fd;

int alarmEnabled = FALSE;
int alarmCount = 0;

int Ns = 0;
int Nr = 1;
```

```
struct termios oldtio;
struct termios newtio;

// Alarm function handler
void alarmHandler(int signal)
{
    alarmEnabled = FALSE;
    alarmCount++;
    printf("Alarm #%d\n", alarmCount);
}

int sendcontrol(unsigned char frame2, unsigned char frame3)
{
    unsigned char frame[5] = {0};
    frame[0] = FLAG;
    frame[1] = frame2;
    frame[2] = frame3;
    frame[3] = frame2 ^ frame3;
    frame[4] = FLAG;
    for (int i = 0; i < 5; i++)
        printf("-%x-", frame[i]);
    write(fd, frame, 5);
    return 0;
}

int setup(const char *serialPortName)
{
    fd = open(serialPortName, O_RDWR | O_NOCTTY);

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

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

    // Clear struct for new port settings
    memset(&newtio, 0, sizeof(newtio));

    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;

    // Set input mode (non-canonical, no echo,...)
    newtio.c_lflag = 0;
    newtio.c_cc[VTIME] = 5; // Inter-unsigned character timer unused
    newtio.c_cc[VMIN] = 0;  // Blocking read until 5 unsigned chars received

    // VTIME e VMIN should be changed in order to protect with a
    // timeout the reception of the following unsigned character(sCLOC)

    // Now clean the line and activate the settings for the port
    // tcflush() discards data written to the object referred to
    // by fd but not transmitted, or data received but not read,
    // depending on the value of queue_selector:
    //   TCIFLUSH - flushes data received but not read.
    tcflush(fd, TCIOFLUSH);

    // Set new port settings
```

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

    printf("New termios structure set\n");
    return 0;
}

int llopen(const char *porta, enum Status status)
{

    printf("\n\n LLOPEN status %i \n\n", status);

    if (setup(porta) < 0)
        return -1;

    enum rec_status state_mach_rec = Start;

    unsigned char byte = 0;
    bool pr = (status == RECEIVER);
    printf("\n bool:%i \n", pr);

    switch (status)
    {
    case RECEIVER:
        printf("\n\nentrou recetor LLOPEN\n\n");

        while (state_mach_rec != STOP)
        {
            if (read(fd, &byte, 1) > 0)
            {
                switch (state_mach_rec)
                {
                case Start:
                    if (byte == FLAG)
                    {
                        state_mach_rec = flag_rcv;
                    }
                    // else {state_mach_rec = Start;}
                    break;
                case flag_rcv:
                    if (byte == A_TRANSMITTER)
                    {
                        state_mach_rec = a_rcv;
                    }
                    else if (byte == FLAG)
                    {
                        state_mach_rec = flag_rcv;
                    }
                    else
                    {
                        state_mach_rec = Start;
                    }
                    break;
                case a_rcv:
                    if (byte == SET)
                    {
                        state_mach_rec = c_rcv;
                    }
                    else if (byte == FLAG)
                    {
                        state_mach_rec = flag_rcv;
                    }
                    else
```

```
                else
                {
                    state_mach_rec = Start;
                }
                break;
            case c_rcv:
                if (byte == (A_TRANSMITTER ^ SET))
                {
                    state_mach_rec = bcc_ok;
                }
                else if (byte == FLAG)
                {
                    state_mach_rec = flag_rcv;
                }
                else
                {
                    state_mach_rec = Start;
                }
                break;
            case bcc_ok:
                if (byte == FLAG)
                {
                    state_mach_rec = STOP;
                }
                else
                {
                    state_mach_rec = Start;
                }
                printf("\nbyte: %i\n", state_mach_rec);
                break;
            default:
                break;
            }
        }
        printf("loop");
    }
    printf("\nmandou receiver\n");
    sendcontrol(A_RECEIVER, UA);
    break;

case TRANSMITTER:
    printf("\n\n Entrou trasmiter LLOPEN \n\n");
    /* code */
    (void)signal(SIGALRM, alarmHandler);

    enum rec_status state_mach_tx = Start;

    alarmCount = 0;

    while (alarmCount < numretransmitions && state_mach_tx != STOP)
    {

        sendcontrol(A_TRANSMITTER, SET);
        alarm(delay); // Set alarm to be triggered in 3s
        alarmEnabled = TRUE;

        // so volta aqui passado 3 segundos

        while (alarmEnabled == TRUE && state_mach_tx != STOP)
        {
            if (read(fd, &byte, 1) > 0)
            {

                switch (state_mach_tx)
                {
                case Start:
```

```
                if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                // else {state_mach_tx = Start;}
                break;
            case flag_rcv:
                if (byte == A_RECEIVER)
                {
                    state_mach_tx = a_tx;
                }
                else if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            case a_tx:
                if (byte == UA)
                {
                    state_mach_tx = c_tx;
                }
                else if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            case c_tx:
                if (byte == (A_RECEIVER ^ UA))
                {
                    state_mach_tx = bcc_ok;
                }
                else if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            case bcc_ok:

                if (byte == FLAG)
                {
                    printf("\nWILL STOP\n");
                    state_mach_tx = STOP;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            default:
                break;
            }
        }
    }
    alarm(0);
```

```
            alarm(0);
        }
        printf("\nCHECKPOINT#1\n");
        alarmCount = 0;

        break;
    default:
        break;
    }

    return fd;
}

// argumentos –fd: identificador da ligação de dados retorno –valor positivo em caso de sucesso –valor negativo em caso de erro
int llclose(int fd)
{

    unsigned char byte;
    enum rec_status state_mach_tx = Start;

    alarmCount = 0;

    //sleep(1);
    while (alarmCount < numretransmitions && state_mach_tx != STOP)
    {

        sendcontrol(A_TRANSMITTER, DISC);
        alarm(delay); // Set alarm to be triggered in 3s
        alarmEnabled = TRUE;

        // so volta aqui passado 3 segundos

        printf("\n llclose recebe:\n");
        while (alarmEnabled == TRUE && state_mach_tx != STOP)
        {
            if (read(fd, &byte, 1) > 0)
            {

                printf("-%x-", byte);

                switch (state_mach_tx)
                {
                case Start:
                    if (byte == FLAG)
                    {
                        state_mach_tx = flag_rcv;
                    }
                    // else {state_mach_tx = Start;}
                    break;
                case flag_rcv:
                    if (byte == A_RECEIVER)
                    {
                        state_mach_tx = a_rcv;
                    }
                    else if (byte == FLAG)
                    {
                        state_mach_tx = flag_rcv;
                    }
                    else
                    {
                        state_mach_tx = Start;
                    }
                    break;
                case a_rcv:
                    if (byte == DISC)
                    {
```

```
                    state_mach_tx = c_rcv;
                }
                else if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            case c_rcv:
                if (byte == (A_RECEIVER ^ DISC))
                {
                    state_mach_tx = bcc_ok;
                }
                else if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            case bcc_ok:
                if (byte == FLAG)
                {
                    state_mach_tx = STOP;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            default:
                break;
            }
        }
    }
    alarm(0);
}

alarmCount = 0;

if (state_mach_tx != STOP)
    return -1;
printf("\n SENDING UA\n");
//sleep(1);
sendcontrol(A_TRANSMITTER, UA);

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

return 0;
}

// return –number of writer unsigned characters –negative value in case of error
int llwrite(int fd, unsigned char *buffer, int length)
{
    int frame_len = length + 6;
```

```
unsigned char *frame = (unsigned char *)malloc(frame_len * sizeof(unsigned char));

frame[0] = FLAG;
frame[1] = A_TRANSMITTER;
frame[2] = C_I(Ns);
frame[3] = frame[1] ^ frame[2];

unsigned char bcc2 = buffer[0];
for (int i = 1; i < length; i++)
{
    bcc2 ^= buffer[i];
}

// memccpy(frame+4,buffer,length);

int pointer_f = 4;

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

    if (buffer[i] == FLAG)
    {
        frame_len++;
        frame = realloc(frame, frame_len);
        frame[pointer_f++] = ESC;
        frame[pointer_f++] = FLAG_R;
        continue;
    }

    else if (buffer[i] == ESC)
    {
        frame_len++;
        frame = realloc(frame, frame_len);
        frame[pointer_f++] = ESC;
        frame[pointer_f++] = ESC_R;
        continue;
    }

    else
    {

        frame[pointer_f++] = buffer[i];
    }
}

// frame[pointer_f++] = bcc2;

//
if (bcc2 == FLAG)
{
    frame_len++;
    frame = realloc(frame, frame_len);
    frame[pointer_f++] = ESC;
    frame[pointer_f++] = FLAG_R;
}

else if (bcc2 == ESC)
{
    frame_len++;
    frame = realloc(frame, frame_len);
    frame[pointer_f++] = ESC;
    frame[pointer_f++] = ESC_R;
}

else
{
```

```
{
    frame[pointer_f++] = bcc2;
}
//

/*

if (frame[pointer_f] == FLAG){
    frame_len++;
    frame = realloc(frame, frame_len);
    frame[pointer_f++] = ESC;
    frame[pointer_f++] = FLAG_R;

}

*/

frame[pointer_f] = FLAG;

enum rec_status state_mach_tx = Start;
bool accepted = 0;
bool rejected = 0;
unsigned char byte = 0;
unsigned char byte_c = 0;

int curr_retransmition = 0;

printf("emissor envia: \n");
for (int i = 0; i < frame_len; i++)
{
    printf("-%x-", frame[i]);
}

printf("\n\n\n");
printf("emissor recebe: \n");
while (curr_retransmition < numretransmitions && state_mach_tx != STOP)
{

    write(fd, frame, frame_len);
    // sendcontrol(A_TRANSMITTER, SET);
    alarm(delay); // Set alarm to be triggered in 3s
    alarmEnabled = TRUE;

    // so volta aqui passado 3 segundos

    while (alarmEnabled == TRUE && state_mach_tx != STOP)
    {
        if (read(fd, &byte, 1) > 0)
        {
            printf("-%x-", byte);

            switch (state_mach_tx)
            {
            case Start:
                if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                // else {state_mach_tx = Start;}
                break;
            case flag_rcv:
                if (byte == A_RECEIVER)
                {
                    state_mach_tx = a_tx;
                }
                else if (byte == FLAG)
```

```
                {
                    state_mach_tx = flag_rcv;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            case a_tx: // new
                byte_c = byte;
                if (byte == RR0 || byte == RR1)
                {
                    state_mach_tx = c_tx;
                    accepted = 1;
                    Ns = (Ns + 1) % 2;
                }
                else if (byte == REJ0 || byte == REJ1)
                {
                    state_mach_tx = c_tx;
                    rejected = 1;
                }
                else if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            case c_tx:
                if (byte == (A_RECEIVER ^ byte_c))
                {
                    state_mach_tx = bcc_ok;
                }
                else if (byte == FLAG)
                {
                    state_mach_tx = flag_rcv;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            case bcc_ok:
                if (byte == FLAG)
                {
                    state_mach_tx = STOP;
                }
                else
                {
                    state_mach_tx = Start;
                }
                break;
            default:
                break;
            }
        }
    }

    alarm(0);
    curr_retransmition++;
    if (rejected && state_mach_tx == STOP)
    {
        printf("rejected");
        rejected = 0;
```

```
            rejected = 0;
            state_mach_tx = Start;
            continue;
        }
        if (accepted && state_mach_tx == STOP)
        {
            printf("accepted");
            break;
        }
        rejected = 0;
        accepted = 0;

    } // 001  0 1010

    alarmCount = 0;
    printf("\n FRAME FREE \n");
    free(frame);

    if (accepted > 0)
        return frame_len;
    else
    {
        printf("\n not accepted \n");
        // llclose(fd);
        return -1;
    }
}

// –fd:       identificador da ligação de dados –buffer: array de caracteres recebidos
// return –array length (number of unsigned characters read) –negative value in case of error
int llread(int fd, unsigned char *buffer)
{

    enum rec_status state_mach_rec = Start;
    unsigned char byte;
    unsigned char c_byte;

    int i = 0;



    printf("\n\n STARTING LLREAD: \n\n");
    while (state_mach_rec != STOP)
    {
        //printf("\n\nINSIDE READ WHILE:: \n\n");
        if (read(fd, &byte, 1) > 0)
        {
            printf("-%x-", byte);
            switch (state_mach_rec)
            {
            case Start:
                if (byte == FLAG)
                {
                    state_mach_rec = flag_rcv;
                }
                // else {state_mach_rec = Start;}
                break;
            case flag_rcv:
                if (byte == A_TRANSMITTER)
                {
                    state_mach_rec = a_rcv;
                }
                else if (byte == FLAG)
                {
                    state_mach_rec = flag_rcv;
                }
```

```
        else
        {
            state_mach_rec = Start;
        }
        break;
    case a_rcv:
        if (byte == C_I(0) || byte == C_I(1))
        {
            state_mach_rec = c_rcv;
            c_byte = byte;
        }
        else if (byte == C_DISC)
        { ////////////// aqui
            c_byte = C_DISC;
            state_mach_rec = c_rcv;
            printf(" \n FOI DETETADO NO READ ANTES \n");
        }
        else if (byte == UA)
        {
            printf("\n\n /////UA RECEIVED /////\n\n");
            c_byte = UA;
            state_mach_rec = c_rcv;
        }
        else if (byte == FLAG)
        {
            state_mach_rec = flag_rcv;
        }
        else
        {
            state_mach_rec = Start;
        }
        break;
    case c_rcv:
        if (byte == (A_TRANSMITTER ^ c_byte) && c_byte == DISC)
        {
            state_mach_rec = bcc_ok;
            printf("\n está certo !!!!! \n");
        }
        else if (byte == (A_TRANSMITTER ^ c_byte) && c_byte == UA)
        {
            state_mach_rec = bcc_ok;
        }

        else if (byte == (A_TRANSMITTER ^ c_byte))
        {
            state_mach_rec = read_data;
        } // pode ler a informacao

        else if (byte == FLAG)
        {
            state_mach_rec = flag_rcv;
        }
        else
        {
            state_mach_rec = Start;
        }
        break;

    case read_data:
        if (byte == ESC)
            state_mach_rec = next_esc;
        else if (byte == FLAG)
        {
            unsigned char bcc2 = buffer[i - 1];
```

```
                buffer[--i] = '\0';

                printf("\n buffer está: \n");
                for (int z = 0; z < i; z++)
                {
                    printf("-%x-", buffer[z]);
                }

                unsigned char bcc_try = buffer[0];

                for (int j = 1; j < i; j++)
                {
                    bcc_try ^= buffer[j];
                }

                printf("recetor envia: \n");
                if (bcc_try == bcc2)
                {
                    state_mach_rec = STOP;
                    sendcontrol(A_RECEIVER, C_RR(Nr));
                    Nr = (Nr + 1) % 2;
                    return i;
                }

                else
                {
                    sendcontrol(A_RECEIVER, C_REJ(Nr));
                    return -1;
                }
            }
            else
            {
                buffer[i++] = byte;
            }

            break;

        case next_esc:
            printf("esc exception: %x\n", byte);
            state_mach_rec = read_data;
            if (byte == FLAG_R)
                buffer[i++] = FLAG;
            else if (byte == ESC_R)
                buffer[i++] = ESC;
            break;

        case bcc_ok:
            printf("\nno bcc ok está: %x \n ", byte);
            if (byte == FLAG)
            {
                state_mach_rec = STOP;
            }

            break;

        default:
            break;
        }
    }
}

if (c_byte == C_DISC)
{
    printf("\n READ SENT DISC TO LLCLOSE \n");
    sendcontrol(A_RECEIVER, C_DISC);
}
```

```
    }

    if (c_byte == UA)
    {

        sleep(1);

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

        printf("\n\n//////////CLOSING/////////////// \n\n");

        close(fd);
        return 0;
    }

    return -1;
}
```

application_layer.h

```
#ifndef _APPLICATION_LAYER_H_
#define _APPLICATION_LAYER_H_

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "link_layer.h"  // Replace with the actual header file you need


#define MAX_PAYLOAD_SIZE 40



//cria um pacote de controlo
unsigned char *MakeCPacket(unsigned char control, unsigned char *filename, long int length, unsigned int *size);
//cria um pacote de dados
unsigned char *MakeDPacket(unsigned char control, unsigned char *data, int length, unsigned int *size);

#endif
```

application_layer.c

```
#include "application_layer.h"

unsigned char *MakeCPacket(unsigned char control, unsigned char *filename, long int length, unsigned int *size)
{
    printf("\nCHECKPOINT#5\n");
    printf("\n LENGTH: %ld\n", length);
    int L1 = (int)ceil(log2f((float)length) / 8.0); // L1 numero de bytes necessário para representar o numero length em hexadecimal
    int L2 = strlen(filename);
    printf("\nSIZE_L1:%d\n", L1);
    printf("\nSIZE_L2:%d\n", L2);
    int sizP = 1 + 2 + L1 + 2 + L2;
    *size = sizP;
    printf("\nSIZE_P:%d\n", sizP);
    unsigned char *packet = (unsigned char *)malloc(sizP);
    int pos = 0;
    packet[pos++] = control;
```

```
    packet[pos++] = 0;
    packet[pos++] = L1;

    // 2 length

    for (int i = 0; i < L1; i++)
    {
        packet[2 + L1 - i] = length & 0xFF;
        length >>= 8;
    }

    pos += L1;

    packet[pos++] = 1;

    packet[pos++] = L2;
    printf("\nCHECKPOINT#7\n");
    memcpy(packet + pos, filename, L2);
    return packet;
}

unsigned char *MakeDPacket(unsigned char control, unsigned char *data, int length, unsigned int *size)
{
    *size = 1 + 2 + length;
    unsigned char *packet = (unsigned char *)malloc(*size);

    packet[0] = 1;
    packet[1] = (length >> 8) & 0xFF;
    packet[2] = length & 0xFF;
    memcpy(packet + 3, data, length);

    return packet;
}




int main(int argc, unsigned char *argv[])
{

    if (argc < 3)
    {
        printf("Incorrect program usage\n"
               "Usage: %s <SerialPort> <device>\n"
               "Example: %s /dev/ttyS1 recetor\n",
               argv[0],
               argv[0]);
        exit(1);
    }

    // unsigned char * delt = (unsigned char *) malloc(2);
    // delt[0] = 'a';
    // delt[1] = 'b';

    // printf("\n\n  %s \n\n",delt);

    const unsigned char *serialPortName = argv[1];
    const unsigned char *status_ = argv[2];
    unsigned char *filename = "pinguim.gif";

    enum Status device = TRANSMITTER;

    if (strcmp(status_, "recetor") == 0)
    {
```

```
{
    printf("\n\nMAIN STRCMP RECEIVER\n\n");
    device = RECEIVER;
}
else if (strcmp(status_, "emissor") == 0)
{
    printf("\n\nMAIN STRCMP TRANSMITTER\n\n");
    device = TRANSMITTER;
}

int fd = llopen(serialPortName, device);
printf("\nCHECKPOINT#2\n");
if (fd < 0)
{
    perror("Connection error\n");
    exit(-1);
}
printf("\nCHECKPOINT#3\n");

double cpu_time_used;
clock_t start, end;


switch (device)
{
case TRANSMITTER:

    FILE *file = fopen(filename, "rb");
    if (file == NULL)
    {
        perror("\nFile not found\n");
        exit(-1);
    }

    long int fileSize0 = ftell(file);
    fseek(file, 0L, SEEK_END);
    long int fileSize = ftell(file) - fileSize0;
    fseek(file, 0L, SEEK_SET);
    unsigned int Psize;
    printf("\nFILE_SIZE: %ld \n", fileSize);
    unsigned char *controlPacketStart = MakeCPacket(2, filename, fileSize, &Psize);
    printf("\n//////////\n//////////\n//////////\nPACKET START WILL WRITE\n//////////\n//////////\n//////////");


    if (llwrite(fd, controlPacketStart, Psize) == -1)
    {
        printf("Exit: error in start packet\n");
        exit(-1);
    }

    printf("\n PACKET_CONTROL FREE \n");
    free(controlPacketStart);

    unsigned char sequence = 0;
    unsigned char *file_content = (unsigned char *)malloc(sizeof(unsigned char) * fileSize);
    fread(file_content, sizeof(unsigned char), fileSize, file);
    long int bytesLeft = fileSize;

    // need to free file_content

    while (bytesLeft >= 0)
    {

        int dataSize = bytesLeft > (long int)MAX_PAYLOAD_SIZE ? MAX_PAYLOAD_SIZE : bytesLeft;
        unsigned char *data = (unsigned char *)malloc(dataSize);
        // need to free data
```

```
            memcpy(data, file_content, dataSize);

            int packetSize;
            unsigned char *packet = MakeDPacket(sequence, data, dataSize, &packetSize);
            // need to free packet;

            if (llwrite(fd, packet, packetSize) == -1)
            {
                printf("Exit: error in data packets\n");
                exit(-1);
            }
            //sleep(1);

            bytesLeft -= (long int)MAX_PAYLOAD_SIZE;
            file_content += dataSize;
            // sequence = (sequence + 1) % 255;
        }

        unsigned int Psize2;
        unsigned char *controlPacketEnd = MakeCPacket(3, filename, fileSize, &Psize2);
        printf("\n//////////\n//////////\n//////////\nPACKET END WILL WRITE\n//////////\n//////////\n//////////");

        if (llwrite(fd, controlPacketEnd, Psize2) == -1)
        {
            printf("Exit: error in exit packet\n");
            exit(-1);
        }

        llclose(fd);

        break;

    case RECEIVER:
        start = clock();
        printf("\n\nRECEIVER#1\n\n");
        unsigned char *packet_r = (unsigned char *)malloc(MAX_PAYLOAD_SIZE);
        int packet_rSize = 0;
        while ((packet_rSize = llread(fd, packet_r)) < 0)
            ;
        printf("\nAAAAAAAAAAAAAA\nAAAAAAAAAAA\nAAAAAAAAAAAAAA\n->CONTROL PACKET: %d\n", packet_r[0]);
        unsigned long int newFileSize = 0;

        // processar o packet recebido

        unsigned char FileSize_NBytes = packet_r[2];

        // unsigned char aux[FileSize_NBytes];
        unsigned char *aux = (unsigned char *)malloc(FileSize_NBytes);

        memcpy(aux, packet_r + 3, FileSize_NBytes);
        printf("\n\nRECEIVER#2\n\n");
        for (unsigned int i = 0; i < FileSize_NBytes; i++)
            newFileSize |= (aux[FileSize_NBytes - i - 1] << (8 * i));
        // tamanho feito

        unsigned char Name_NBytes = packet_r[3 + FileSize_NBytes + 1];
        unsigned char *nome = (unsigned char *)malloc(Name_NBytes + 9);
        memcpy(nome, packet_r + 3 + FileSize_NBytes + 2, Name_NBytes);
        memcpy(nome + Name_NBytes - 4, "_novo.gif", 9);
        printf("\nNOME:%s\n", nome);
        // nome feito

        FILE *newFile = fopen((unsigned char *)nome, "wb+");
        printf("\n\nRECEIVER#3\n\n");
```

```
        while (1)
        {

            // stuck
            while ((packet_rSize = llread(fd, packet_r)) < 0)
            {
            }
            //printf("\nREAD PACKET\n");
            printf("\n//////////\n//////////\n//////////\nPACKET 0: %d\n//////////\n//////////\n//////////", packet_r[0]);
            if (packet_rSize == 0){
                printf("Packet size is 0\n");
                break;
            }
            else if (packet_r[0] != 3)
            {
                unsigned char *buffer = (unsigned char *)malloc(packet_rSize);

                memcpy(buffer, packet_r + 3, packet_rSize - 3);
                //buffer += packet_rSize + 4;

                fwrite(buffer, sizeof(unsigned char), packet_rSize - 3, newFile);
                printf("\n BUFFER FREE \n");
                free(buffer);
            }
            else
            {
                printf("\n//////////\n//////////\n//////////\nBREAK PACKET %d\n//////////\n//////////\n//////////", packet_r[0]);
                printf("\n\nRECEIVER WHILE\n\n");
                continue;
            }
        }

        fclose(newFile);
        end = clock;
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        break;

    default:
        exit(-1);
        break;
    }
    printf("\nCHECKPOINT##\n");
    return 0;
}
```