Faculdade de Engenharia da Universidade do Porto

1º trabalho laboratorial – Protocolo de Ligação de Dados

Relatório

Turma 11

João Tomás Marques Félix up202008867

Miguel Lourenço Pregal de Mesquita Montes up202007516

Sumário

Este relatório incidirá sobre o primeiro trabalho proposto na unidade curricular de Redes de Computadores cujo principal objetivo foi o desenvolvimento de uma aplicação que fosse capaz de transmitir ficheiros entre dois computadores de forma assíncrona através de uma porta de série.

A aplicação é capaz de transferir corretamente de um computador para o outro.

Introdução

O objetivo do relatório é examinar a componente teórica do trabalho realizado, cujo objetivo foi a implementação de um protocolo de ligação de dados entre dois computadores usando uma porta de série para ser possível a transferência de informação entre eles.

O relatório está organizado da seguinte forma:

- **Arquitetura** – blocos funcionais e interfaces;
- **Estrutura do código** – APIs, principais estruturas de dados, principais funções e a sua relação com a arquitetura;
- **Casos de uso principais** – identificação, sequências de chamada de funções;
- **Protocolo da ligação lógica** – identificação dos principais aspetos funcionais, descrição da estratégia de implementação destes aspetos com apresentação de extratos de código;
- **Protocolo de aplicação** – identificação, sequências de chamada de funções
- **Validação** – descrição dos testes efetuados com apresentação quantificada dos resultados, se possível;
- **Eficiência do protocolo de ligação de dados** - caraterização estatística da eficiência do protocolo, efetuada recorrendo a medidas sobre o código desenvolvido. A caraterização teórica de um protocolo *Stop&Wait*, que deverá ser empregue como termo de comparação, encontra-se descrita nos slides de Ligação Lógica das aulas teóricas;
- **Conclusões** - síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados;

Arquitetura

A aplicação desenvolvida consiste em duas camadas, a *Link Layer* e a *Application Layer*.

Link Layer

Esta camada encontra-se no ficheiro **link_layer.c** e contém as funções necessárias para iniciar e terminar a ligação entre os dois computadores através da porta de série, bem como ler e escrever.

Application Layer

Esta camada encontra-se no ficheiro **application_layer.c** e serve como ponte entre o utilizador e a *link layer*, recebendo os argumentos do utilizador e tratando de tudo o que seja relacionado com abertura, leitura e escrita de e para ficheiros.

Estrutura do código

O código do programa está dividido em 2 ficheiros, consoante as camadas *application layer* ou link layer. A cada um destes está também associado um ficheiro *header*. O programa inclui ainda um ficheiro macros.h que define as macros usadas no programa.

Application Layer – application_layer.c

Contém todas as funções e estruturas de dados mais gerais da *Application Layer*.

- **controlPacket:** cria e armazena em memória o pacote de controlo;
- **dataPacket:** cria e armazena em memória o pacote de dados;
- **compareCPacket:** comparação entre os dados enviados e recebidos relativamente ao pacote de controlo;
- **applicationLayer:** Realiza a abertura do ficheiro. Processa as tramas de controlo. Contém o ciclo de envio de tramas de informação, invoca a função **llwrite** para envio das tramas de dados. Contém também o ciclo de leitura de tramas de informação, invoca a função **llread**;

Data Link Layer – link_layer.c

- **state_machine:** Armazena os campos de interesse para a máquina de estados e retorna o estado atual;
- **buildFrame:** constrói a trama;
- **sendAck e sendNack:** funções para realizar *Stop&Wait ARQ*;
- **llopen:** Abre a porta de série e troca as tramas SET e UA;
- **llread:** Lê as tramas I e envia a correspondente trama de supervisão (RR ou REJ);
- **llwrite:** Envia as tramas I;
- **llclose:** Efetua a troca de tramas DISC e UA e fecha a ligação à porta de série;

Macros – macros.h

Este ficheiro contém as macros usadas no programa.

Casos de uso

A aplicação deve ser compilada executando o comando *make clean* e de seguida *make*. Para simular o cabo fazer *sudo ./bin/cable* e em seguida seguir as instruções do ficheiro *README* a partir do ponto 4, que nos vão permitir correr o transmissor e o recetor e no final comparar o ficheiro enviado e o ficheiro recebido e verificar se são iguais.

Podemos, por exemplo, abrir 4 terminais e em cada um deles executar o *sudo ./bin/cable*, *make run_rx*, *make run_tx* e *make check_files*, respetivamente.

O recetor deverá ser iniciado em primeiro lugar. Caso contrário, a aplicação do transmissor irá enviar alarmes por ainda não ter conseguido estabelecer ligação e após esse tempo, se a aplicação do recetor ainda não tiver iniciado, irá terminar o programa.

Após o correto início das duas aplicações, o transmissor irá tratar de abrir e recolher a informação sobre o ficheiro a ser enviado e chamar a função *llwrite* para enviar essa informação para o recetor. Por sua vez, o *recetor* irá, através do *llread*, obter a informação e escreve para o novo ficheiro.

Protocolo de ligação lógica

O protocolo de ligação lógica permite a configuração inicial e a terminação da ligação da porta de série, o envio e a leitura de informação da porta.

Utilizando para isso funções auxiliares que ajudam tanto na construção das mensagens para envio como nas rotinas de leitura e escrita para a prevenção de erros.

As seguintes funções foram implementadas:

llopen()

Esta função tem o propósito de iniciar a comunicação entre os dois computadores utilizando a porta de série.

Começa por chamar a função open, abrindo a porta de série com as *flags* de leitura e escrita. É então configurada a porta com o VTIME a 0 e o VMIN a 0, para que a função *read* não esteja à espera de um caractere antes de retornar.

Consoante o *role* passado como argumento, irá comparar e verificar se é transmissor ou recetor.

A primeira função, do transmissor, irá enviar a mensagem SET e esperar por uma mensagem UA. Esta função irá então construir a mensagem, enviá-la, e fazer a leitura para receber o UA que só retorna caso tenha lido ou caso tenha ocorrido um erro após 3 tentativas de envio e leitura.

A segunda função, do recetor, tenta ler o valor recebido e mostra qual o estado atual da *state machine*. Posteriormente cria e envia a trama UA.

lwrite()

Esta função permite o envio de pacotes de dados pela porta de série.

É responsável por construir a trama de informação, após isso, trata de enviar o pacote e esperar por uma resposta do tipo RR ou REJ consoante o pacote tenha sido, ou não, corretamente enviado e recebido pelo recetor.

llread()

Esta função permite a leitura de pacotes de dados pela porta de série.

A função lê uma trama de informação. Após isso, faz *destuffing* da mensagem e verifica o BCC2 para certificar que a mensagem recebida não contém erros. Caso o BCC2 calculado dos

dados seja diferente do BCC2 recebido, é enviada a mensagem do tipo REJ, é descartada a mensagem. Caso esteja tudo correto, é guardada a mensagem e é enviada uma resposta do tipo RR.

llclose()

Esta função tem o propósito de terminar a ligação da porta de série entre os dois computadores.

Esta função, consoante o *role*, irá invocar o transmissor ou o recetor.

A primeira função, do transmissor, irá enviar a mensagem DISC e esperar por uma mensagem DISC. Após esta receção, é enviada a mensagem UA para que o recetor saiba que a conexão foi terminada.

A segunda função, do recetor, trata de ler uma mensagem do tipo DISC, envia uma mensagem do tipo DISC e espera por uma mensagem do tipo UA.

Depois disto, são repostas as configurações iniciais da porta e é terminada a ligação chamando a função *close*.

Protocolo de aplicação

O protocolo de aplicação é responsável pela abertura e leitura do ficheiro a enviar, escrita para o ficheiro de destino, construção e envio de pacotes de controlo e dados.

controlPacket()

Esta função permite a construção de um pacote de controlo e o seu armazenamento em memória.

dataPacket()

Esta função permite a construção de um pacote de dados através da leitura do ficheiro recebido do transmissor.

compareCPacket()

Esta função faz a verificação do pacote recebido e retorna erro caso encontre algum problema.

applicationLayer()

Esta função permite encapsular, caso seja recetor ou transmissor, as rotinas que devem executar para que seja possível a transferência/receção correta do ficheiro pretendido.

Caso seja transmissor começa por abrir o ficheiro a enviar e a obter o seu tamanho. Após isso, inicia a construção do pacote de dados e procede ao envio do ficheiro e ao seu consequente fecho. No meio deste processo faz as verificações de erros para perceber se tudo é enviado corretamente.

No caso de ser recetor cria um ficheiro com o nome adequado e prepara-se para guardar informação no mesmo. Recebe a informação, verifica se está correta, e caso esteja,

retorna o ficheiro recebido corretamente. Por fim encerra a escrita no ficheiro e caso não consiga efetuar o close retorna um erro.

Validação

Foram efetuados diversos testes à nossa aplicação:

- **Envio de ficheiros de diversos tamanhos** - penguin.gif (10968 bytes), pesado.jpg (158607 bytes)
- **Envio do ficheiro com pacotes de diferentes tamanhos** - 32, 64, 128, 256, 512, 1024, 2048, 4096;
- **Envio do ficheiro com valores diferentes de Baudrate** - 9600, 19400, 38000, 59600, 115000;
- **Envio do ficheiro com diferentes *timeouts*** – 1s, 2s, 3s, 5s, 10s, 20s, 30s;
- **Envio do ficheiro com interrupção no cabo;**
- **Primeiro iniciar a transferência e só passado algum tempo iniciar a receção;**

Eficiência do protocolo de ligação de dados

Todos os testes efetuados foram realizados nos computadores dos laboratórios, para que fossem obtidos resultados viáveis.

Variação do Baudrate

Para os valores testados, não foram observadas quaisquer alterações significativas na eficiência do programa.

Variação do tempo

Com o aumento do tempo, a decadência da eficiência do protocolo é também exponencial pois o aumento do tempo faz com que cada mensagem demore mais a ser enviada, tanto como de dados como respostas

Conclusões

Este trabalho consistia na implementação de um protocolo de ligação de dados para comunicação através de uma porta de série. Um dos principais objetivos do trabalho era a compreensão de que deveriam ser criadas duas camadas. Cada uma com uma responsabilidade atribuída, e independentes entre si, tal que o modo de funcionamento de uma em nada deve influenciar o modo de funcionamento da outra.

Ao realizar este trabalho foi clara a distinção destas camadas, a *link layer* e a *application layer*, sendo a primeira responsável pela comunicação através da porta de série, tratando da construção das mensagens, *stuffing,* erros, respostas, *etc,* enquanto a *application* apenas utiliza as funções desta camada, mas sem o conhecimento de nenhum destes mecanismos de tratamento de erros, ou de qualquer outro tipo de funcionamento da camada.

O trabalho foi concluído com sucesso, cumprindo os objetivos preconizados e contribuindo para um aprofundamento do conhecimento sobre os mecanismos de envio de informação e, mais concretamente, da interface com uma porta de série.

Anexo I – Código Fonte

macros.h

```c
// MISC
#define _POSIX_SOURCE 1 // POSIX compliant source
#define BUF_SIZE 256 //tamanho do buf
#define MAX_DATA_SIZE (MAX_PAYLOAD_SIZE+4)

#define FLAG   0x7E
#define FLAG_P 0x7D //escape; PPP mechanism

#define A_WRITE 0x03
#define A_READ  0x01

//Control
#define C_UA    0x07
#define C_SET   0x03
#define C_DISC  0x0B
#define C_RR0   0x05
#define C_RR1   0x85
#define C_REJ0  0x81
#define C_REJ1  0x01

#define BCC(x,y) (x^y)

#define FALSE 0
#define TRUE  1

//numeracao da trama de info
#define I0 0x00 // NS = 0
#define I1 0x40 // NS = 1
```

application_layer.h

```c
// Application layer protocol header.
// NOTE: This file must not be changed.

#ifndef _APPLICATION_LAYER_H_
#define _APPLICATION_LAYER_H_

// Application layer main function.
// Arguments:
//   serialPort: Serial port name (e.g., /dev/ttyS0).
//   role: Application role {"tx", "rx"}.
//   baudrate: Baudrate of the serial port.
//   nTries: Maximum number of frame retries.
//   timeout: Frame timeout.
//   filename: Name of the file to send / receive.
void applicationLayer(const char *serialPort, const char *role, int baudRate,
                      int nTries, int timeout, const char *filename);

#endif // _APPLICATION_LAYER_H_
```

link_layer.h

```c
// Link layer header.
// NOTE: This file must not be changed.

#ifndef _LINK_LAYER_H_
#define _LINK_LAYER_H_

typedef enum
{
    LlTx,
    LlRx,
} LinkLayerRole;

typedef struct
{
    const char *serialPort;
    LinkLayerRole role;
    int baudRate;
    int nRetransmissions;
    int timeout;
} LinkLayer;

// SIZE of maximum acceptable payload.
// Maximum number of bytes that application layer should send to link layer
#define MAX_PAYLOAD_SIZE 1000

// MISC
#define FALSE 0
#define TRUE 1

// Open a connection using the "port" parameters defined in struct linkLayer.
// Return "1" on success or "-1" on error.
int llopen(LinkLayer connectionParameters);

// Send data in buf with size bufSize.
// Return number of chars written, or "-1" on error.
int llwrite(const unsigned char *buf, int bufSize);

// Receive data in packet.
// Return number of chars read, or "-1" on error.
int llread(unsigned char *packet);

// Close previously opened connection.
// if showStatistics == TRUE, link layer should print statistics in the console on close.
// Return "1" on success or "-1" on error.
int llclose(int showStatistics);

#endif // _LINK_LAYER_H_
```

main.c

```c
// Main file of the serial port project.
// NOTE: This file must not be changed.

#include <stdio.h>
#include <stdlib.h>

#include "application_layer.h"

#define BAUDRATE 9600
#define N_TRIES 3
#define TIMEOUT 4

// Arguments:
//   $1: /dev/ttySxx
//   $2: tx | rx
//   $3: filename
int main(int argc, char *argv[])
{
    if (argc < 4)
    {
        printf("Usage: %s /dev/ttySxx tx|rx filename\n", argv[0]);
        exit(1);
    }

    const char *serialPort = argv[1];
    const char *role = argv[2];
    const char *filename = argv[3];

    printf("Starting link-layer protocol application\n"
           "  - Serial port: %s\n"
           "  - Role: %s\n"
           "  - Baudrate: %d\n"
           "  - Number of tries: %d\n"
           "  - Timeout: %d\n"
           "  - Filename: %s\n",
           serialPort,
           role,
           BAUDRATE,
           N_TRIES,
           TIMEOUT,
           filename);

    applicationLayer(serialPort, role, BAUDRATE, N_TRIES, TIMEOUT, filename);

    return 0;
}
```

application_layer.c

src > application_layer.c > ...

```c
#include "../include/application_layer.h"
#include "../include/link_layer.h"
#include "../include/macros.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

extern int reject;

long int filesize(FILE *file){
    fseek(file, 0, SEEK_END); //pointer no final do ficheiro

    long int bytesSize = ftell(file); //pos atual do pointer

    fseek(file, 0, SEEK_SET); //pointer no inicio

    return bytesSize;
}

int hexBytes(long int bytes){
    int hex = 0;

    while(bytes != 0){
        bytes >>= 8;
        hex++;
    }
    return hex;
}

//Pacote de controlo como no slide 23
void controlPacket(unsigned char *CPacket, int filesizeHexSize, long int bytesSize, unsigned int size, const char *filename){
    CPacket[0] = 2; //start
    CPacket[1] = 0; //tamanho do ficheiro
    CPacket[2] = filesizeHexSize; //tamanho hexadecimal

    memcpy(&CPacket[3], &bytesSize, filesizeHexSize);//alocado em memoria

    CPacket[3+filesizeHexSize] = 1; //nome do ficheiro
    CPacket[3+filesizeHexSize+1] = size; //tamanho nome ficheiro

    memcpy(&CPacket[3+filesizeHexSize+2], filename, size);
}

//Pacote de dados como no slide 23
void dataPacket(unsigned char *data_packet, int size, unsigned int n, FILE *file_Tx){
    unsigned char buf[size];

    fread(buf, 1, size, file_Tx);

    data_packet[1] = n; //dados
    data_packet[2] = size / 256;
    data_packet[3] = size % 256;

    for (int i=0; i<size; i++)
        data_packet[i+4] = buf[i];
}

int compareCPacket(unsigned char *packet, int filesizeHexSize, unsigned char filesizeHex[MAX_PAYLOAD_SIZE], unsigned char receiv
    if (packet[1] == 0) { // tipo do parâmetro->file_size_bytes (em string)
```

src > C application_layer.c > ...

```c
int compareCPacket(unsigned char *packet, int filesizeHexSize, unsigned char filesizeHex[MAX_PAYLOAD_SIZE], unsigned char receivedFilename[MAX_PAYLOAD_SIZE], unsigned char size){
    if (packet[1] == 0) { // tipo do parâmetro->file_size_bytes (em string)
        if(filesizeHexSize != packet[2]){
            printf("Start and end are different\n"); //
            return -1;
        }

        for(int i = 0; i < filesizeHexSize; i++){
            if(filesizeHex[i] != packet[3+i]){
                printf("Start and end are different\n");
                return -1;
            }
        }
    }

    if (packet[3+filesizeHexSize] == 1){ // tipo do parâmetro->filename
        if(size != packet[3+filesizeHexSize+1]){
            printf("Start and end are different\n");
            return -1;
        }

        for(int i = 0; i < size; i++){
            if(receivedFilename[i] != packet[3+filesizeHexSize+2+i]){
                printf("Start and end are different\n");
                return -1;
            }
        }
    }

    return 0;
}

void applicationLayer(const char *serialPort, const char *role, int baudRate, int nTries, int timeout, const char *filename)  //try to open files, send and receive them
{
    LinkLayer linkLayer;
    linkLayer.serialPort = serialPort; //porta de serie

    if (!strcmp(role, "tx"))
        linkLayer.role = LlTx;

    else if (!strcmp(role, "rx"))
        linkLayer.role = LlRx;

    linkLayer.baudRate = baudRate; //velocidade
    linkLayer.nRetransmissions = nTries; //tentativas
    linkLayer.timeout = timeout; //valor do temporizador

    if(llopen(linkLayer) == -1){
        printf("Failed llopen\n");
        return;
    }

    if(linkLayer.role == LlTx){ //todo o processo de envio de ficheiro

        FILE *file_Tx;
        file_Tx = fopen(filename, "r"); //ler

        if (file_Tx == NULL) {  //no file
            printf("Failed to open file\n");
```

src > C application_layer.c > ...

```c
        if (file_Tx == NULL) {  //no file
            printf("Failed to open file\n");
            return;
        }

        long int bytesSize = filesize(file_Tx);

        int filesizeHexSize = hexBytes(bytesSize);

        unsigned int size = strlen(filename); //length

        unsigned int CPacket_size = 3 + filesizeHexSize + 2 + size; // 3->C1 T1 L1; 2->T2 L2; length(V1) = file_size_bytes; length(V2) = size

        unsigned char CPacket[CPacket_size];

        controlPacket(CPacket, filesizeHexSize, bytesSize, size, filename);

        if (llwrite(CPacket, CPacket_size) == -1){
            printf("Failed llwrite\n");
            return;
        }


        //iniciar construção do pacote de dados
        unsigned char data_packet[MAX_PAYLOAD_SIZE+4] = {0};
        data_packet[0] = 1; //C; valor 1 = dados
        unsigned int n = 0;
        int last_packet = FALSE;

        while (!last_packet){

            if (bytesSize > MAX_DATA_SIZE){
                bytesSize -= MAX_DATA_SIZE;
                size = MAX_DATA_SIZE;
            }

            else{
                last_packet = TRUE;
                size = bytesSize;
            }

            dataPacket(data_packet, size, n, file_Tx);

            n = (n+1)%255;
            int counter = 0;

            reject = FALSE;

            //se "while" só em vez de "do while" ele recebe um txt vazio em vez de gif
            do {
                if(llwrite(data_packet, (int) (size+4)) == -1){
                    printf("llwrite failed\n");
                    return;
                }
                counter++;
            } while (reject && counter < nTries);
        }
        fclose(file_Tx); //close file
```

src > C application_layer.c > ...

```c
        fclose(file_Tx); //close file

        //end control packet
        CPacket[0] = 3; //3 equivale a end

        if(llwrite(CPacket, CPacket_size) == -1){ //escrita falhou
            printf("Failed llwrite\n");
            return;
        }

        printf("\n");
        printf("File sent\n");
        printf("\n");
    }

    else if(linkLayer.role == LlRx) //todo o processo de receção de ficheiro
    {

        FILE *file_Rx;
        file_Rx = fopen(filename, "w+"); //para ler e escrever

        if (file_Rx == NULL) { //dont have a file
            printf("Failed to open file\n");
            return;
        }

        unsigned char buf[MAX_PAYLOAD_SIZE]={0};

        if(llread(buf) == -1){ //can't read control packet
            printf("Control packet failed\n");
            return;
        }

        unsigned char filesizeHexSize = 0;
        unsigned char filenameSize;
        unsigned char filesizeHex[MAX_PAYLOAD_SIZE];
        unsigned char receivedFilename[MAX_PAYLOAD_SIZE];
        int num_seq = 0;

        //=2 equivale  start
        if(buf[0] == 2){

            if (buf[1] == 0) { //tamanho do ficheiro
                filesizeHexSize = buf[2]; //L1

                for(int i = 0; i < filesizeHexSize; i++){
                    filesizeHex[i] = buf[3+i];
                }
            }

            if (buf[3+filesizeHexSize] == 1){ //nome do ficheiro
                filenameSize = buf[3+filesizeHexSize+1];

                for(int i = 0; i < filenameSize; i++){
                    receivedFilename[i] = buf[3+filesizeHexSize+2+i];
                }
            }


```

src > C application_layer.c > ...

```c

            while(TRUE){

                unsigned char packet[MAX_PAYLOAD_SIZE]={0};

                if(llread(packet) == -1){
                    printf("Failed to read packet\n");
                    return;
                }

                // read end control packet
                if(packet[0] == 3){
                    if (compareCPacket(packet, filesizeHexSize, filesizeHex, receivedFilename, filenameSize)<0)
                        return;
                    break;
                }

                //de acordo com slide 23
                //=1 valor de dados
                if(packet[0] == 1){
                    unsigned char n, l1, l2;
                    unsigned int k;
                    n = packet[1];

                    if(n != num_seq){
                        printf("Failed to read packet\n");
                        continue;
                    }

                    num_seq = (num_seq+1)%255;
                    l2 = packet[2];
                    l1 = packet[3];
                    k = 256*l2 + l1;
                    unsigned char data[k];

                    for(int i = 0; i < k; i++){ //campo de dados do pacote
                        data[i] = packet[4+i];
                    }

                    fwrite(data, 1, k, file_Rx);
                }

            }
        }
        fclose(file_Rx);
        printf("\n");
        printf("File received\n");
        printf("\n");
    }

    if(llclose(0) == -1){
        printf("Failed llclose\n");
    }
}
```

link_layer.c

src > C link_layer.c > ...

```c
#include "../include/link_layer.h"
#include "../include/macros.h"
#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 <signal.h>

unsigned char ns = 0, nr = 1; //ns começa a 0 e nr a 1
unsigned char control = 0; //campo de controlo

unsigned int tentativas;
unsigned int timeout;
char role;
int fd;

int reject = FALSE;

struct termios oldtio, newtio;

typedef enum {
    START,
    FLAG_RCV,
    A_RCV,
    C_RCV,
    BCC_OK,
} State; //trama

int alarmEnabled = FALSE;
int alarmCount = 0;

// Alarm functions

void alarmHandler(int signal)
{
    alarmEnabled = FALSE;
    alarmCount++;

    printf("Alarm #%d\n", alarmCount);
}

void activateAlarm(){ //start alarm
    alarm(timeout); //Temporizador
    alarmEnabled = TRUE;
}

void deactivateAlarm(){ //end alarm
    alarm(0);
    alarmEnabled = FALSE;
    alarmCount = 0;
}

//Fases do protocolo de Ligação de Dados

void ns_update(){ //change value I
```

src > C link_layer.c > ...

```c

void ns_update(){ //change value I
    if(ns == I0){ //anterior 0
        ns = I1; //novo 1
    } else { //se anterior 1
        ns = I0; // novo 0
    }
}

void nr_update(){ //change value RR
    if(nr == 0){ //anterior 0
        nr = 1; //novo 1
    } else { //se anterior 1
        nr = 0; //novo 0
    }
}

//Maquina de estados para construção de trama
State state_machine(State current_state, int *finished, unsigned char value, unsigned char a, unsigned char c){
    switch (current_state){
        case START:
            if(value == FLAG){
                current_state = FLAG_RCV; //flag
            }
            break;
        case FLAG_RCV:
            if(value == FLAG){
                return current_state; //retorna estado atual
            }
            else if(value == a){
                current_state = A_RCV; //A
            }
            else
                current_state = START; //inicio
            break;
        case A_RCV:
            if(value == FLAG){
                current_state = FLAG_RCV; //flag
            }
            else if(value == c){
                current_state = C_RCV; //C
            }
            else{
                current_state = START; //inicio
            }
            break;
        case C_RCV:
            if(value == FLAG){
                current_state = FLAG_RCV; //flag
            }
            else if(value == (a^c)){
                current_state = BCC_OK; //BCC
            }
            else{
                if(role == LlRx){
                    printf("Error");
                }
                current_state = START; //volta ao inicio
            }
```

src > C link_layer.c > ...

```
115                     current_state = START; //volta ao inicio
116                 }
117                 break;
118             case BCC_OK:
119                 if(value == FLAG){
120                     *finished = TRUE; //fim
121                 }
122                 else{
123                     current_state = START; //inicio
124                 }
125                 break;
126         }
127         return current_state; //retorna estado atual
128     }
129
130     //Trama
131     unsigned int buildFrame(unsigned char* updated_I, unsigned int I_size, const unsigned char *buf, int bufSize){
132         updated_I[0] = FLAG; //F
133         updated_I[1] = A_WRITE; //A
134         unsigned char c;
135
136         if(ns == 0){ //valor de I(NS=0)
137             c = I0;
138         } else { //valor de I(NS=1)
139             c = I1;
140         }
141
142         updated_I[2] = c; //C
143         updated_I[3] = BCC(A_WRITE,c); //BCC1
144
145         //Dados
146         int bcc2 = 0; //xor
147         int start = 4;
148         int extra = 0; // 0 como nos exemplos para acrescentar algo
149
150         //Transparência - Mecanismo de byte stuffing
151
152         for(int i = 0; i < bufSize; i++) {
153
154             //Se ocorrer 0x7e então 0x7d 0x5e
155             if(buf[i] == FLAG){
156                 updated_I[i+start+extra] = FLAG_P;
157                 updated_I[i+start+1+extra] = 0x5E;
158                 extra++;
159             }
160
161             else if(buf[i] == FLAG_P){ //Se ocorrer 0x7d então 0x7d 0x5d
162                 updated_I[i+start+extra] = FLAG_P;
163                 updated_I[i+start+1+extra] = 0x5D;
164                 extra++;
165             }
166
167             else{
168                 updated_I[i+start+extra] = buf[i];
169             }
170             bcc2 ^= buf[i];
171         }
172
```

src > C link_layer.c > ...

```c
    //BCC2
    if(bcc2 == FLAG){ //Se ocorrer 0x7e então 0x7d 0x5e
        updated_I[I_size-3] = FLAG_P;
        updated_I[I_size-2] = 0x5E;
    }

    else if(bcc2 == FLAG_P){ //Se ocorrer 0x7d então 0x7d 0x5d
        updated_I[I_size-3] = FLAG_P;
        updated_I[I_size-2] = 0x5D;
    }

    else{

        I_size--;
        updated_I[I_size-2] = bcc2;
    }
    updated_I[I_size-1] = FLAG;
    return I_size;
}

//Stop and Wait ARQ (slide 31, data-link-layer)
int sendAck(){
    unsigned char ack[5];
    ack[0] = FLAG;
    ack[1] = A_WRITE;
    ack[4] = FLAG;

    if (nr == 0) {
        ack[2] = C_RR0;
        ack[3] = BCC(A_WRITE, C_RR0);
    } else if (nr == 1) {
        ack[2] = C_RR1;
        ack[3] = BCC(A_WRITE, C_RR1);
    }

    if(write(fd,ack,5) == -1){
        printf("failed writing\n");
        return -1;
    }

    return 0;
}

int sendNack(){
    unsigned char nack[5];
    nack[0] = FLAG;
    nack[1] = A_WRITE;
    nack[4] = FLAG;

    if(nr == 0){
        nack[2] = C_REJ0;
        nack[3] = BCC(A_WRITE, C_REJ0);
    }

    else if(nr == 1){
        nack[2] = C_REJ1;
        nack[3] = BCC(A_WRITE, C_REJ1);
    }

```

src > C link_layer.c > ...

```c

    if(write(fd,nack,5) == -1){
        printf("failed writing\n");
        return -1;
    }

    return 0;
}

////////////////////////////////////////////////
// LLOPEN
////////////////////////////////////////////////
int llopen(LinkLayer connectionParameters) //identificador da ligação de dados
{
    tentativas = connectionParameters.nRetransmissions;
    timeout = connectionParameters.timeout;
    role = connectionParameters.role;

    //like write_noncanonical.c

    fd = open(connectionParameters.serialPort, O_RDWR | O_NOCTTY);

    if (fd < 0){
        return -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 = connectionParameters.baudRate | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;

    // Set input mode (non-canonical, no echo,...)
    newtio.c_lflag = 0;
    newtio.c_cc[VTIME] = 0; // Inter-character timer unused
    newtio.c_cc[VMIN] = 0;  // Blocking read until 5 chars received
    // VTIME e VMIN should be changed in order to protect with a
    // timeout the reception of the following character(s)

    tcflush(fd, TCIOFLUSH);

    // Set new port settings
    if (tcsetattr(fd, TCSANOW, &newtio) == -1)
    {
        perror("tcsetattr");
        exit(-1);
    }

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

    if(role == LlTx){ //sender
```

src > C link_layer.c > ...

```c
if(role == LlTx){ //sender
    unsigned char set_value[5]; //inicia trama
    set_value[0] = FLAG;
    set_value[1] = A_WRITE;
    set_value[2] = C_SET;
    set_value[3] = BCC(A_WRITE,C_SET);
    set_value[4] = FLAG;

    int finished = FALSE;

    (void)signal(SIGALRM, alarmHandler); //liga alarme
    State current_state = START; //estado

    while(!finished && alarmCount < tentativas)
    {
        (void)signal(SIGALRM, alarmHandler);
        if (!alarmEnabled)
        {
            current_state = START;
            if (write(fd, set_value, 5) == -1){
                perror("Couldn't write\n");
                return -1;
            }
            activateAlarm();
        }

        unsigned char value;
        int bytes = read(fd, &value, 1); //lê

        if(bytes == -1){
            perror("Failed to read1");
            return -1;
        }

        current_state = state_machine(current_state, &finished, value, A_WRITE, C_UA);
    }
    deactivateAlarm();
}

else if(role == LlRx){ //receiver
    State current_state = START;
    int finished = FALSE;

    while(!finished){
        unsigned char value;

        int bytes = read(fd, &value, 1); //lê

        if(bytes == -1){
            perror("Couldn't read");
            return -1;
        }
        current_state = state_machine(current_state, &finished, value, A_WRITE, C_SET);
    }

    unsigned char ua[5];
    ua[0] = FLAG;
    ua[1] = A_WRITE;
    ua[2] = C_UA;
```

src > C link_layer.c > ...

```c
        ua[2] = C_UA;
        ua[3] = BCC(A_WRITE, C_UA);
        ua[4] = FLAG;

        if (write(fd, ua, 5) == -1){
            perror("Couldn't write\n");
            return -1;
        }
    }
    return fd;
}


////////////////////////////////////////////////
// LLWRITE
////////////////////////////////////////////////
int llwrite(const unsigned char *buf, int bufSize) //número de caracteres escritos
{
    reject = FALSE;
    deactivateAlarm();

    int count = 0;
    for (int i = 0; i < bufSize; i++) {
        if (buf[i] == FLAG || buf[i] == FLAG_P) {
            count += 1;
        }
    }

    unsigned int I_size = 6 + bufSize + count + 1;

    unsigned char updated_I[I_size];

    //Trama de informação
    I_size = buildFrame(updated_I, I_size, buf, bufSize);

    int finished = FALSE;
    (void) signal(SIGALRM, alarmHandler);
    State current_state = START;

    alarmEnabled = FALSE;

    while (!finished && alarmCount < tentativas) {
        if (!alarmEnabled){
            if(write(fd, updated_I, I_size) == -1){ //tenta escrever trama de informação
                perror("Couldn't write\n");
                return -1;
            }
            current_state = START; //inicio

            activateAlarm();
        }

        unsigned char value;
        int bytes = read(fd, &value, 1); //lê

        if(bytes == -1){
            perror("Couldn't read");
            continue;
        }
```

src > C link_layer.c > ...

```
405         }
406
407         switch (current_state){ //Trama S + uma maquina de estados para leitura
408             case START:
409                 if(value == FLAG){
410                     current_state = FLAG_RCV;
411                 }
412                 break;
413             case FLAG_RCV:
414                 if(value == FLAG){
415                     continue;
416                 }
417                 else if(value == A_WRITE){
418                     current_state = A_RCV;
419                 }
420                 else{
421                     current_state = START;
422                 }
423                 break;
424             case A_RCV:
425                 if(value == FLAG){
426                     current_state = FLAG_RCV;
427                 }
428                 else if(value == C_RR0){
429                     if (ns==0)
430                         reject = TRUE;
431                     else{
432                         control = value;
433                         current_state = C_RCV;
434                         reject = FALSE;
435                     }
436                 } else if(value == C_RR1){
437                     if (ns==1)
438                         reject = TRUE;
439                     else{
440                         control = value;
441                         current_state = C_RCV;
442                         reject = FALSE;
443                     }
444                 }
445                 else if (value == C_REJ0 || value == C_REJ1){
446                     read(fd, &value, 1);
447                     reject = TRUE;
448                     return 0;
449                 }
450                 else{
451                     current_state = START;
452                 }
453                 break;
454             case C_RCV:
455                 if(value == FLAG){
456                     current_state = FLAG_RCV;
457                 }
458                 else if(value == (BCC(A_WRITE,control))){
459                     current_state = BCC_OK;
460                 }
461                 else{
462                     current_state = START;
463                 }
```

src > C link_layer.c > ...

```
                }
                break;
            case BCC_OK:
                if(value == FLAG){
                    finished = TRUE;
                }
                else{
                    current_state = START;
                }
                break;
        }
    }

    if(!finished){
        printf("I can not wait anymore\n");
        printf("Nothing received\n");
        return -1;
    }
    ns_update();

    deactivateAlarm();

    return I_size;
}

////////////////////////////////////////////////
// LLREAD
////////////////////////////////////////////////
int llread(unsigned char *packet) //comprimento do array (número de caracteres lidos)
{
    int finished = FALSE;
    (void)signal(SIGALRM, alarmHandler);
    State current_state = START;

    int PPP_on = FALSE; //Flag P on
    int bcc2 = 0;
    int size_data = 0;
    char c;
    unsigned char value;

    while(!finished) {
        int bytes = read(fd, &value, 1); //lê

        if (bytes == 0){
            continue;
        }
        if(bytes == -1){
            perror("Failed to read2");
            return -1;
        }

        switch (current_state){ //leitura da trama I
            case START:
                if(value == FLAG){
                    current_state = FLAG_RCV;
                }
                break;
            case FLAG_RCV:
                if(value == FLAG){
```

src > C link_layer.c > ...

```
520                 case FLAG_RCV:
521                     if(value == FLAG){
522                         continue;
523                     }
524                     else if(value == A_WRITE){
525                         current_state = A_RCV;
526                     }
527                     else{
528                         current_state = START;
529                     }
530                     break;
531                 case A_RCV:
532                     if(value == FLAG){
533                         current_state = FLAG_RCV;
534                     }
535                     else if(value == I0) {
536                         if (nr == 0) {  //Discards duplicate
537                             if (sendNack() == -1)
538                                 return -1;
539                             current_state = START;
540                         }
541                         else {
542                             c = value;
543                             current_state = C_RCV;
544                         }
545                     }else if (value == I1){
546                         if (nr == 1) {  //Discards duplicate
547                             if (sendNack() == -1)
548                                 return -1;
549                             current_state = START;
550                         }
551                         else {
552                             c = value;
553                             current_state = C_RCV;
554                         }
555                     }
556                     break;
557                 case C_RCV:
558                     if(value == FLAG){
559                         current_state = FLAG_RCV;
560                     }
561                     else if(value == (BCC(A_WRITE,c))){ // BCC1
562                         current_state = BCC_OK;
563                     }
564                     else{
565                         printf("Failed\n");
566                         current_state = START;
567                     }
568                     break;
569 
570                 case BCC_OK:
571                     if(value != FLAG){
572                         //Destuffing
573                         if(value == FLAG_P){
574                             PPP_on = TRUE;
575                             continue;
576                         }
577 
578                         if(PPP_on){
579                             if(value == 0x5E){
```

src > C link_layer.c > ...

```c
                    if(PPP_on){
                        if(value == 0x5E){
                            packet[size_data] = FLAG;
                            size_data++;
                        }
                        else if(value == 0x5D){
                            packet[size_data] = FLAG_P;
                            size_data++;
                        }
                        PPP_on = FALSE;
                    }

                    else {
                        packet[size_data] = value;
                        size_data++;
                    }
                }
                else{
                    finished = TRUE;
                }
                current_state = BCC_OK;
                break;
            }
        }

        bcc2 = packet[size_data-1];
        size_data--;
        unsigned char new_bcc2 = 0;

        for (int i=0; i<size_data; i++) {
            new_bcc2 ^= packet[i];

        }

        if (new_bcc2 == bcc2) { //Positive answer
            if (sendAck() == -1)
                return -1;
        }

        else {
            printf("Failed\n"); //Negative answer
            if (sendNack() == -1)
                return -1;
        }

        nr_update();
        return size_data;
    }

    ////////////////////////////////////////////////
    // LLCLOSE
    ////////////////////////////////////////////////
    int llclose(int showStatistics) //valor positivo em caso de sucesso e negativo em caso de erro
    {
        deactivateAlarm();

        if(role == LlTx){
            unsigned char disc_value[5];
```

src > C link_layer.c > ...

```c
        unsigned char disc_value[5];
        disc_value[0] = FLAG;
        disc_value[1] = A_READ;
        disc_value[2] = C_DISC;
        disc_value[3] = BCC(A_READ,C_DISC);
        disc_value[4] = FLAG;

        int finished = FALSE;
        (void) signal(SIGALRM, alarmHandler);
        State current_state = START;

        while(!finished && alarmCount < tentativas )
        {
            if (!alarmEnabled)
            {
                current_state = START;
                if (write(fd, disc_value, 5) == -1){
                    perror("Couldn't write\n");
                    return -1;
                }
                activateAlarm();
            }

            unsigned char value;

            int bytes = read(fd, &value, 1);

            if(bytes == -1){
                perror("Couldn't read\n");
                return -1;
            }

            current_state = state_machine(current_state, &finished, value, A_READ, C_DISC);
        }
        deactivateAlarm();

        unsigned char ua_value[5];
        ua_value[0] = FLAG;
        ua_value[1] = A_READ;
        ua_value[2] = C_UA;
        ua_value[3] = BCC(A_READ,C_UA);
        ua_value[4] = FLAG;

        if (write(fd, ua_value, 5) == -1){
            perror("Couldn't write\n");
            return -1;
        }

    }

    else if(role == LlRx){
        State current_state = START;
        int finished = FALSE;

        while(!finished){
            unsigned char value;

            int bytes = read(fd, &value, 1);
```

src > C link_layer.c > llclose(int)

```
693
694             if(bytes == -1){
695                 perror("Failed to read3\n");
696                 return -1;
697             }
698             current_state = state_machine(current_state, &finished, value, A_READ, C_DISC);
699         }
700
701         unsigned char disc[5];
702         disc[0] = FLAG;
703         disc[1] = A_READ;
704         disc[2] = C_DISC;
705         disc[3] = BCC(A_READ, C_DISC);
706         disc[4] = FLAG;
707
708         if (write(fd, disc, 5) == -1){
709             perror("Couldn't write\n");
710             return -1;
711         }
712
713         current_state = START;
714         finished = FALSE;
715
716         while(!finished){
717             unsigned char value;
718
719             int bytes = read(fd, &value, 1);
720
721             if(bytes == -1){
722                 perror("Failed to read4\n");
723                 return -1;
724             }
725             current_state = state_machine(current_state, &finished, value, A_READ, C_UA);
726         }
727     }
728
729     if (tcsetattr(fd, TCSANOW, &oldtio) == -1) {
730         perror("tcsetattr");
731         exit(-1);
732     }
733
734     close(fd); //close file
735
736     return 1;
737 }
```