Ligação de Dados

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Grupo:
José Carlos Alves Vieira - up201404446
Luis Figueiredo Caseiro - up201306125
Renato Sampaio de Abreu - up201403377

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

3 de Novembro de 2016

Índice

1. Sumário

Este projeto, desenvolvido no contexto da cadeira Redes de Computadores do Mestrado Integrado de Engenharia Informática, tem como objetivo a implementação de um protocolo de ligação de dados e o teste desse protocolo usando uma aplicação simples de transferência de dados entre dois computadores, recorrendo para isso às respectivas portas série de cada um. Posto isto, o projeto incide maioritariamente em conteúdos que tínhamos abordado durante as aulas teóricas, tais como a Physical Layer e Data Link Layer.

O relatório permitiu-nos aprofundar os aspectos mais teóricos dos conteúdos anteriormente referidos, de forma a relacionar a parte teórica com a parte prática, possibilitando uma melhor aprendizagem por parte do grupo e ao mesmo tempo facilitar o processo de correção do projeto por parte do docente.

2. Introdução

O objectivo proposto para este primeiro trabalho de Redes de Computadores era a implementação de um protocolo de ligação de dados de acordo com a especificação fornecida (serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão, no nosso caso, pela porta de série). Também era objectivo o teste desse protocolo com uma aplicação simples de transferência de dados, usando comunicação assíncrona.

Usou-se uma variante *Stop and Wait* durante a implementação. Garantiu-se transmissão de dados independente de códigos (transparência) assegurada pela técnica de byte stuffing. A transmissão é organizada em tramas que podem ser de 3 tipos: informação (I), Supervisão (S) e Não Numeradas (U). Essas tramas são protegidas por um código detector de erros.

Este relatório subdivide-se nas seguintes secções:

- **Introdução**, onde é realizada uma introdução ao objectivo do projecto.
- **Arquitetura**, onde são apresentados os blocos funcionais e as interfaces.
- **Estrutura do Código**, referindo as APIs implementadas, as principais estruturas de dados e a sua relação com a arquitetura.
- **Casos de Uso Principais**, fazendo a sua identificação e apresentando as sequências de chamada de funções.
- **Protocolo de Ligação Lógica**, onde se identifica os principais aspectos funcionais, as estratégias de implementação desses mesmos aspectos (por vezes com extratos de código).
- **Protocolo de Aplicação**, onde se identifica os principais aspectos funcionais, as estratégias de implementação desses mesmos aspectos (por vezes com extratos de código).
- **Validação**, descrevendo os testes efectuados com a apresentação dos resultados nesses testes.
- **Elementos de Valorização**, onde se indicam os elementos de valorização implementados e quais as estratégias elaboradas para o conseguir.
- **Conclusões**, elaborando uma síntese das secções apresentadas e uma breve reflexão sobre os objectivos académicos alcançados.

3. Arquitetura

O projeto desenvolvido está estruturado em duas camadas: aplicação, especificado nos ficheiros ApplicationLayer.h e ApplicationLayer.c, e ligação de dados, presente nos ficheiros LinkLayer.h e LinkLayer.h. Com esta estrutura, ambas as camadas não conhecem os detalhes inerentes a cada uma.

Assim, a camada de aplicação desconhece as especificações e funções - formato de tramas, transparência, retransmissões, etc - da ligação de dados, mas é a partir dela que são acedidos os serviços presentes na ligação de dados. Esta camada atua de forma diferente de acordo com o modo do programa, Transmitter ou Receiver. De uma forma muito geral, o emissor é responsável pela leitura da informação do ficheiro e envio desses dados para a camada de ligação de dados, enquanto que no recetor é realizada a receção de dados e posterior escrita no ficheiro.

Por outro lado, na camada de ligação de dados são disponibilizadas as funções genéricas de ligação de dados tais como: abertura e fecho da ligação, controlo de erros, controlo de fluxo, sincronismo e confirmações positivas/negativas. Desta forma, esta camada não tem acesso aos serviços da aplicação existindo assim uma interdependência entre camadas.

A interface da consola, especificado no ficheiro main.c, oferece ao utilizador um leque variado de opções, as quais este pode decidir alterar ou não. O programa é executado com os valores dados por default, contudo o utilizador tem a possibilidade de definir os valores dos seguintes parâmetros: Baud rate, tamanho máximo do campo de informação das tramas I, número máximo de retransmissões e intervalo de time out. A definição destes parâmetros é aceite desde que os valores introduzidos sejam válidos. Desta forma, o utilizador tem a possibilidade de influenciar diretamente o funcionamento do programa.

4. Estrutura do Código

Principais estruturas de dados

Existem 2 principais estruturas de dados: **ApplicationLayer** e **LinkLayer** (Ver anexo I).

A **ApplicationLayer** possui um descritor de ficheiro, um status (zero no PC emissor, 1 no PC receptor), o nome do ficheiro a enviar, o tamanho do ficheiro a enviar, o tamanho do nome do ficheiro e o tamanho da informação a enviar (pode ser definida pelo utilizador).

Principais Funções da **ApplicationLayer**:

No emissor, existem 3 funções principais que incorporam as funcionalidades mais importantes existentes na ApplicationLayer:

- SendData -> Responsável pela leitura do ficheiro e chamadas às funções de envio de pacotes.
- SendControl -> Responsável pelo envio de pacotes de controlo, start ou end.
- SendInformation -> Responsável pela criação e envio dos pacotes de dados.

No recetor, existem também 3 funções indispensáveis ao funcionamento correto do programa:

- ReceiveData-> Responsável pela escrita no ficheiro e chamadas às funções de receção de pacotes.

- ReceiveControl-> Responsável pela receção de pacotes de controlo, start ou end.
- ReceiveInformation-> Responsável pela receção de pacotes de dados e verificação da sua integridade.

O **LinkLayer** possui o nome da porta de série, a taxa de transmissão de informação, o número de timeouts (pode ser definido pelo utilizador), o número de retransmissões máximas (pode ser definido pelo utilizador), o tamanho da trama (pode ser definida pelo utilizador), o status, o Ns (inicialmente a zero), o número de tramas de rejeição enviadas, a definição atual do controlo da trama I, RR e REJ tendo em conta o Ns e duas estruturas termios (uma para guardar o modo da consola inicial, e outro para guardar o novo modo).

Principais funções da **LinkLayer**:

As 4 principais funções são: llopen, llwrite, llread e llclose. Estas são as funções que permitem estabelecer a ligação com a porta de série, escrever para a porta de série, ler da porta de série e fechar a ligação com a porta de série, respectivamente. Dentro destas funções existem outras como establishConnection, startConnection e endConnection. Todas estas funções são do domínio da camada da ligação de dados.

5. Casos de Uso Principais

O programa desenvolvido é constituído maioritariamente por duas sequências de funcionamento, uma presente ao executar em modo Transmitter e outra no modo Receiver.

Assim, a sequência de ambos os modos está definida, em traços gerais, no seguinte diagrama:

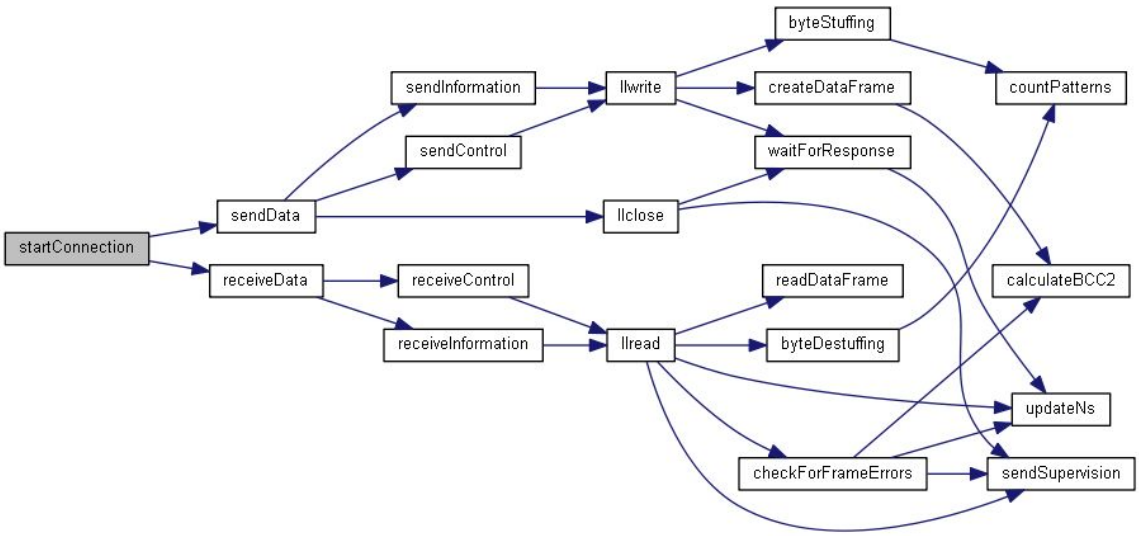

Figura 1 - Diagrama de Sequência

A sequência de leitura do ficheiro e envio dos pacotes - modo Transmitter - consiste na chamada da função sendData, que no início envia o pacote START, atráves de sendControl, e posteriormente inicia o envio de pacote de dados através de sendInformation. Ambas as funções "send" utilizam o serviço llwrite da camada de ligação de dados. Nesta camada, são realizadas as

operações de criação da trama I e envio dessa trama. Antes do retorno à função sendData é chamada a função waitForResponse que espera pela confirmação de receção da trama. Por fim, é enviado o pacote END novamente atráves de sendControl.

A sequência de leitura de dados recebidos e escrita no ficheiro - modo Receiver- consiste na chamada da função receiveData, que no início recebe o pacote START, atráves de receiveControl, e posteriormente inicia a receção do pacote de dados através de receiveInformation. Ambas as funções "receive" utilizam o serviço llread da camada de ligação de dados. Nesta camada, são realizadas as operações de receção da trama I e confirmação dessa trama. Por fim, é recebido o pacote END que sinaliza o fim da transmissão de pacotes de informação.

Assim, é notória a similaridade entre ambas as sequências, sendo neste caso algo de extrema importância para que o programa funcione em total sincronização.

6. Protocolo de Ligação Lógica

A camada applicationLayer depende das implementações da linkLayer. Desta forma, em linkLayer estão implementadas funções críticas para o processamento, envio e receção das tramas de informação do programa.

Esta camada é responsável por estabelecer a comunicação entre o transmissor/receptor e a porta de série, pelo envio das tramas, pela sua leitura e pelo stuffing e destuffing aplicado às tramas.

llopen

Quando é chamada pelo transmissor, envia a trama SET e espera pelo envio da trama UA do receptor. Caso acontece timeout no transmissor (passou 'timeout' segundos sem receber a trama), este retransmite a trama SET e fica novamente à espera de receber a trama UA. Caso o transmissor não consiga receber a trama UA nas primeiras 'numTransmissions' tentativas, a execução do programa é interrompida.

Quando é chamada pelo receptor, este espera até receber a trama SET e envia a trama de confirmação UA.

llWrite

Função usada exclusivamente pelo emissor que recebe como argumento um buffer contendo a informação de um pacote de controlo ou de um pacote de dados. Seguidamente, é chamada a função createDataFrame, adicionando à informação já obtida os bytes que faltavam para completar uma trama I, preenchendo mais 6 posições de bytes (flag, campo de endereço, campo de controlo, BCC, BCC2 e novamente a flag).

Depois realiza o stuffing dos bytes da trama (mecanismo usado para protecção da trama).

Por fim, sempre que ocorre um timeout, continua a enviar a trama I até que consiga ler a resposta do receptor. Caso envie mais vezes que as permitidas, a execução do programa é interrompida. Quando a trama RR é recebida com sucesso, o Ns é atualizado.

llRead

Esta é uma função usada somente pelo emissor que recebe como argumento um package que será preenchido com a informação da trama recebida.

Posto isto, é feita a chamada da função readDataFrame que modificará o buffer enviado com a informação da trama lida. Seguidamente, faz-se o destuffing do buffer, e copia-se apenas a informação com os dados da trama para o package. Desta forma o package apenas terá informação útil para escrita no ficheiro. Por fim, verifica-se se o buffer contém erros. Caso tudo esteja certo, o receptor envia a trama RR para sinalizar o transmissor para avançar. Também é atualizado o Ns.

llClose

Esta função realiza o término da ligação e o fecho da porta de série. É usada pelo emissor e pelo receptor.

No transmissor: envia a trama DISC. De seguida o transmissor fica à espera da confirmação enviada pelo receptor sob a forma da trama DISC. Caso ocorra timeout, o transmissor reenvia tramas DISC até que o receptor envie com sucesso a trama DISC ou o número de tramas DISC enviadas pelo recetor seja maior do que o número de tentativas permitidas. Caso seja capaz de ler com sucesso a trama DISC enviada pelo receptor, envia a trama UA e "dorme" por 1 segundo antes de fechar a ligação com a porta de série.

No receptor: fica à espera de ler com sucesso a trama DISC enviada pelo transmissor. De seguida envia a trama DISC e fica à espera de ler com sucesso a trama UA enviada pelo transmissor.

Seguidamente, tanto o transmissor como o receptor terminam a ligação à porta de série.

7. Protocolo de Aplicação

A applicationLayer é responsável pela invocação das principais funções da linkLayer. É onde ocorre a invocação das funções principais do linkLayer (llopen, llwrite, llread, llclose).

Permite o controlo do tipo de pacotes enviados pela porta de série. Existem 2 tipos: pacotes de controlo e pacotes de dados. Os pacotes de controlo são enviados duas vezes: uma para marcar o início da transferência de informação, e outro para marcar o fim. O pacote de controlo inicial possui um bit de controlo diferente do bit de controlo do pacote final, de forma a definir a sua diferença (0x02 e 0x03, respectivamente).

Existem duas funções para tratar do pacote de controlo: **sendControl** e **receiveControl**.

sendControl: esta função gera um pacote de controlo e preenche-o, inicializando o pacote com o tipo de controlo apropriado (CONTROL_START ou CONTROL_END), tamanho e nome do ficheiro. Em seguida é invocada a função llwrite para o envio do pacote de controlo.

receiveControl: chama a função llread, obtendo o package com o pacote de controlo recebido. É agora verificada informação e guardada na estrutura applicationLayer, informação essa relativa a todo o pacote de controlo: tipo, tamanho do ficheiro e nome do ficheiro.

Existem duas funções para tratar dos pacotes de dados: **sendInformation** e **receiveInformation**.

sendInformation: parecida em estrutura com a função sendControl, esta função gera um pacote de dados e preenche-o com os dados apropriados: o tipo (CONTROL_DATA), o número da trama, o número de octetos L2 e L1 e os bytes que contêm os dados do ficheiro a enviar. De seguida invoca a função llwrite para enviar o pacote de dados ao receptor.

receiveInformation: invoca a função llread, obtendo o package com o pacote de dados recebido. De seguida verifica a informação recebida, como o tipo, o atual número de tramas

recebidas (verificando também problemas de sincronização), e guarda os tamanhos dos octetos. Apenas a informação do ficheiro é guardada num buffer para ser usada posteriormente.

Existem duas funções que interligam o envio de todos os pacotes: **sendData** e **receiveData**.

sendData: abre o ficheiro que se deseja enviar em mode "read binary". Envia o pacote de controlo inicial (**sendControl**), e enquanto não tiver lido todos os bytes do ficheiro, vai enviando os bytes lidos sobre a forma de pacotes de dados (**sendInformation**). Quando todo o ficheiro foi enviado, o mesmo é fechado e é enviado o pacote de controlo final (**sendControl**).

receiveData: fica à espera de receber o pacote de controlo inicial (**receiveControl**). Cria um ficheiro com o nome recebido pelo pacote de controlo em modo "write binary", e até não ter escrito o mesmo número de bytes indicado pelo pacote de controlo, fica a receber pacotes de dados (**receiveInformation**), detectando quando o pacote de dados é repetido (case -2) ou novo (default), escrevendo no ficheiro à medida que vai recebendo dados. Finalmente, tenta receber o pacote de controlo final(**receiveControl**).

8. Validação

De modo a testar e validar o bom funcionamento do programa, inicialmente utilizamos a imagem fornecida no moodle, "pinguim.gif", testando em máquinas virtuais e posteriormente em ambiente real. Aquando dos testes em ambiente real, além da transferência normal que funcionava corretamente, com a existência de ruído ou de paragem na transferência de dados, a imagem era sempre enviada com sucesso entre o emissor e receptor.

Por outro lado, devido à geração aleatória de erros implementada, verificamos que o programa, mesmo sem interferências, procede corretamente em caso de bytes errados o que permite validar o funcionamento das confirmações negativas/positivas (REJ).

Deste modo, e com o objetivo de testar o programa em várias situações, também experimentamos a transferência de um ficheiro .jpg com 1830988 bytes. O resultado foi bastante satisfatório como se pode verificar pela imagem seguinte:

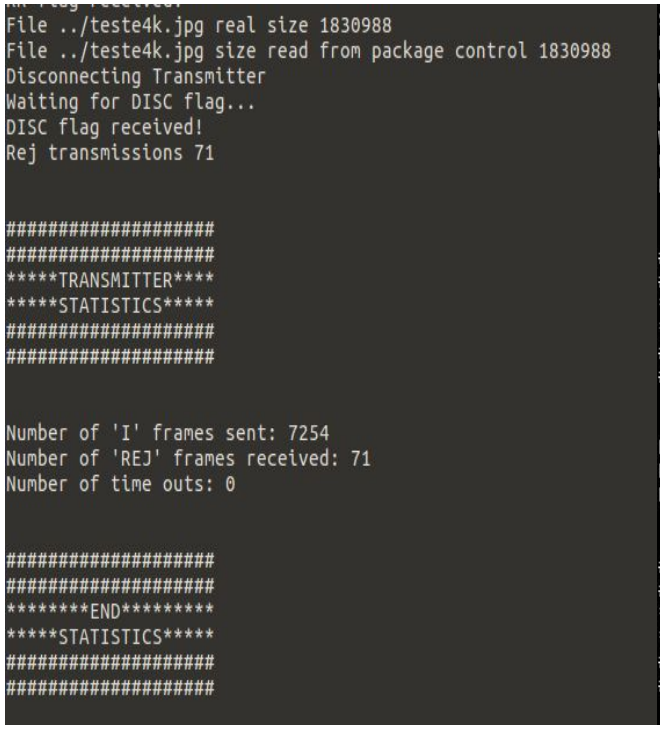

Figura 2 - Emissor

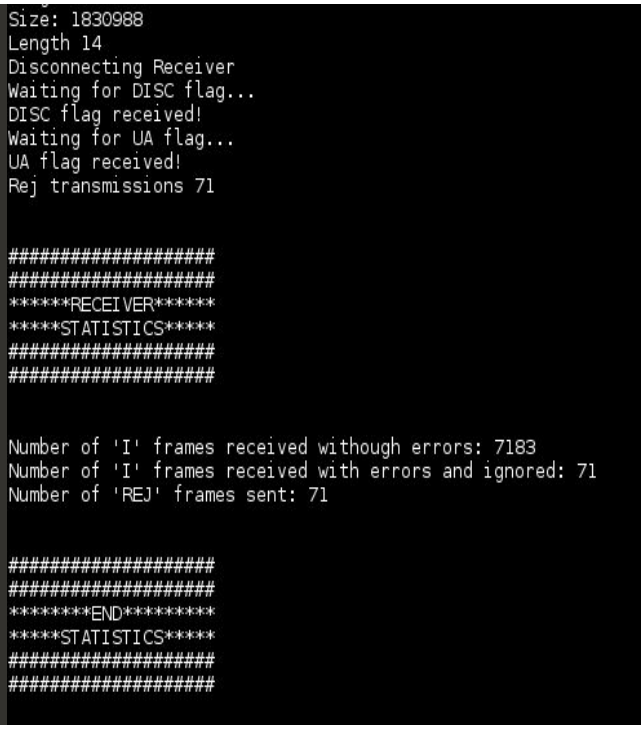

Figura 3 - Recetor

9. Elementos de Valorização

Selecção de parâmetros pelo utilizador.

Quando o utilizador executa o programa, depara-se com uma interface onde é possível mudar o baud rate, o tamanho máximo a transmitir por cada trama, o número máximo de retransmissões e o tempo a esperar antes de ocorrer time out.

Implementação de REJ

Aquando da receção de uma trama pelo receptor, esta é posta à prova através de testes ao seu campo de controlo, ao BCC e ao BCC2. Caso alguma destas condições falhe, o receptor envia a trama REJ para o transmissor.

```c
int checkForFrameErrors(int fd, unsigned char *buffer, unsigned char *package, int length, int dataSize) {
    //Check if Control is wrong (using Ns)
    if(buffer[2] != linkLayer->controlI) {
        linkLayer->numREJ++;

        if((buffer[2] << 7) != linkLayer->controlI)
            updateNs();

        sendSupervision(fd, linkLayer->controlREJ);
        return -1;
    }

    //Checks if BCC1 is wrong
    if(buffer[3] != (buffer[1] ^ buffer[2])) {
        linkLayer->numREJ++;
        sendSupervision(fd, linkLayer->controlREJ);
        printf("Error during BCC1\n");
        return -1;
    }

    //Checks if BCC2 is wrong
    unsigned char bcc2Read = buffer[length - 2];
    unsigned char bcc2FromBytes = calculateBCC2(package, dataSize);

    if(bcc2Read != bcc2FromBytes){
        //printf("Different BCC's\n");
        //printf("bcc2Read = %c, bcc2FromBytes = %c\n", bcc2Read, bcc2FromBytes);
        linkLayer->numREJ++;
        sendSupervision(fd, linkLayer->controlREJ);
        printf("Error during BCC2\n");
        return -1;
    }

    return 0;
}
```

Figura 4 - Verificação de erros na trama I

Verificação da integridade dos dados pela Aplicação

A aplicação verifica o tamanho real do ficheiro, bem como o valor indicado nos pacotes de controlo. Também verifica e descarta tramas duplicadas/perdidas, e existe uma recuperação em caso de erro.

Registo de ocorrências

As ocorrências das diferentes tramas enviadas são registadas ao longo da execução do programa. Sempre que alguma é enviada, o seu contador é incrementado. No final são apresentadas as estatísticas do transmissor e do receptor, cada um com as estatísticas que lhe dizem respeito.

```c
typedef struct LinkLayer {
    char port[20];
    int baudRate;
    unsigned int timeout;
    unsigned int numTransmissions;
    int frameLength;
    int status;
    int ns;
    int numREJ;
```

```c
int numFrameI = 0;
int numFrameItransmitted = 0;
int numTimeOuts = 0;
```

Figura 5 - Contadores das ocorrências

Geração aleatória de erros em tramas I

Tal como explicitado no guião, em cada trama I recebida é simulado no campo de dados com probabilidades definidas e aleatórias (utilização de srand), a alteração de um byte, procedendo-se posteriormente à análise dessa trama de forma normal. Obtém-se assim a rejeição da trama e consequente envio de confirmação negativa (REJ).

```
if(rand() % 100 == 0){
    int byteIndex = (rand() % dataSize) + 4;
    buffer[byteIndex] = ERROR;
}
```

Figura 6 - Geração de erros

10. Conclusões

Todos os objectivos propostos no enunciado foram atingidos com sucesso.

O projecto permitiu aos alunos utilizar uma variante de transmissão de *Stop and Wait* para implementar um protocolo de ligação de dados. Desta forma foi possível realizar uma aprendizagem mais profunda relativamente ao uso da porta de série, máquinas de estados aplicadas a protocolos e protocolos usados em redes, portanto o grupo concluiu que foi conseguido um bom aproveitamento tendo em conta o objectivo proposto.

As camadas foram implementadas de forma a serem independentes entre si e a informação do cabeçalho das tramas apenas faz parte da camada de ligação de dados. A camada da aplicação consegue aceder às funções da camada de ligação de dados, mesmo sem a necessidade de perceber como é que elas funcionam, permitindo que conceitos de mais baixo nível sejam abstratos a esta camada.

Em suma, o desenvolvimento do projecto foi exigente devido às especificidades pretendidas - principalmente a forma de interação e sincronização da camada de aplicação e de dados em ambos os computadores - mas podemos concluir que permitiu consolidar conhecimentos teóricos aprendidos ao longo das aulas bem como perceber como interagem as camadas existentes num protocolo de ligação de dados.

11. Contribuição

José Carlos Alves Vieira - 47.5%
Luis Figueiredo Caseiro - 5%
Renato Sampaio de Abreu - 47.5%

12. Anexo I

Arquivo .zip enviado juntamente com o relatório contendo o código fonte e o doxygen gerado.

13. Anexo II

Principais estruturas de dados

```
typedef struct ApplicationLayer{
    /*Serial port descriptor*/
    int fileDescriptor;
    /*TRANSMITTER | RECEIVER*/
    int status;

    char * fileName;
    unsigned int fileSize;
    int nameLength;
    int dataLength;
}ApplicationLayer;
```

Figura 7 - ApplicationLayer struct

```
int sendData(){ ···
}

int sendControl(int type){ ···
}

int sendInformation(unsigned char * buffer, int length){ ···
}

int receiveData(){ ···
}

int receiveControl(int control){ ···
}

int receiveInformation(unsigned char *buffer, int *length){ ···
}
```

Figura 8 - Principais funções da applicationLayer

```
typedef struct LinkLayer {
    char port[20];
    int baudRate;
    unsigned int timeout;
    unsigned int numTransmissions;
    int frameLength;
    int status;
    int ns;
    int numREJ;
    unsigned int controlI;
    unsigned int controlRR;
    unsigned int controlREJ;
    struct termios oldtio, newtio;
}LinkLayer;
```

Figura 9 - LinkLayer struct

```
int llopen(int fd){ ···
}

int llclose(int fd){ ···
}

int llwrite(unsigned char * buffer, int length, int fd){ ···
}

int llread(int fd, unsigned char *package, int numFrame){ ···
}
```

Figura 10 - Principais funções da LinkLayer

```c
int llopen(int fd){
    srand(time(NULL));
    STOP = FALSE;
    timer = 1;
    if(linkLayer->status == 0){
        printf("%s\n", "Connecting Transmitter");
        int nTry = 1;
        int messageReceived = 0;
        while(nTry <= linkLayer->numTransmissions && !messageReceived){
            sendSupervision(fd, C_SET);
            if(waitForResponse(fd, UA) == -1){
                nTry++;
            } else messageReceived = 1;
        }
        if(!messageReceived){
            printf("%s\n", "Error estabilishing connection!");
            return -1;
        }
    } else if(linkLayer->status == 1){
        printf("%s\n", "Connecting Receiver");
        waitForResponse(fd, SET);
        sendSupervision(fd, C_UA);
    }
    return 1;
}
```

Figura 12 - Função llopen

```c
int llread(int fd, unsigned char *package, int numFrame){
    unsigned char * buffer;
    int length, dataSize;

    while(1) {
        buffer = malloc(MAX_FRAME_LENGTH);

        length = readDataFrame(fd, buffer);
        if(length == -1) {
            printf("Error reading frame, trying again...\n");
            continue;
        }

        printf("Frame length with stuffing: %d\n", length);
        length = byteDestuffing(&buffer, length);
        printf("Frame length after destuff: %d\n",length);

        dataSize = length - 6;
        //printf("Data size = %d\n", dataSize);

        memcpy(package, &buffer[4], dataSize);

        printf("Esta e a frame %d\n", numFrame);
        if(checkForFrameErrors(fd, buffer, package, length, dataSize) == -1) {
            free(buffer);
            continue;
        }

        //Sends RR
        printf("Frame %d enviando RR\n", numFrame);
        sendSupervision(fd, linkLayer->controlRR);
        updateNs();
        free(buffer);
        break;
    }

    return length - 6;
}
```

Figura 13 - Função llread

```c
int llwrite(unsigned char * buffer, int length, int fd){
    int newSize = length + 6;
    unsigned char *frame = createDataFrame(buffer, length);
    int sizeAfterStuff = byteStuffing(&frame, newSize);

    int bytesSent;
    int nTry = 1;
    int messageReceived = 0;
    while(nTry <= linkLayer->numTransmissions && !messageReceived){
        bytesSent = write(fd, frame, sizeAfterStuff);
        numFrameItransmitted++;
        if(bytesSent != sizeAfterStuff){
            printf("%s\n", "Error sending data packet");
            return -1;
        }
        if(waitForResponse(fd, RR) == -1){
            nTry++;
        } else messageReceived = 1;
    }
    if(!messageReceived){
        printf("%s\n", "Error receiving packet confirmation!");
        return -1;
    }
    return 1;
}
```

Figura 14 - Função llwrite

```
int llclose(int fd){
    STOP = FALSE;
    timer = 1;
    if(linkLayer->status == 0){
        printf("%s\n", "Disconnecting Transmitter");
        int nTry = 1;
        int messageReceived = 0;
        while(nTry <= linkLayer->numTransmissions && !messageReceived){
            sendSupervision(fd, C_DISC);
            if(waitForResponse(fd, DISC) == -1){
                nTry++;
            } else messageReceived = 1;
        }
        if(!messageReceived){
            printf("%s\n", "Error terminating connection!");
            return -1;
        }
        sendSupervision(fd, C_UA);
        sleep(1);
    } else if(linkLayer->status == 1){
        printf("%s\n", "Disconnecting Receiver");
        int nTry = 1;
        int messageReceived = 0;
        while(!messageReceived){
            if(waitForResponse(fd, DISC) == -1){
                nTry++;
            } else messageReceived = 1;
        }
        if(!messageReceived){
            printf("%s\n", "Error terminating connection!");
            return -1;
        }
        nTry = 1;
        messageReceived = 0;
        while(nTry <= linkLayer->numTransmissions && !messageReceived){
            sendSupervision(fd, C_DISC);
            if(waitForResponse(fd, UA) == -1){
                nTry++;
            } else messageReceived = 1;
        }
        if(!messageReceived){
            printf("%s\n", "Error terminating connection!");
            return -1;
        }
    }

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

    printf("Rej transmissions %d\n", linkLayer->numREJ);

    return 1;
}
```

Figura 15 - Função llclose