**Protocolo de Ligação de Dados**

(1º Trabalho Laboratorial)



**Mestrado Integrado em Engenharia Informática e Computação**

**EIC0032- Redes de Computadores**

António Manuel Vieira Ramadas 201303568

Rui Filipe Freixo Cardoso Osório 201303843

Rui Miguel Teixeira Vilares 201207046

5 de novembro de 2015

# Sumário

Este trabalho laboratorial foi realizado no âmbito da unidade curricular de Redes e Computadores, do curso Mestrado Integrado em Engenharia Informática e Computação. O objetivo foi transferir um ficheiro entre dois computadores, através de uma porta de série. Entre os conceitos abordados, destacam-se a *Application Layer*, a *Data Link Layer* e a *Physical Layer*. Todos os conceitos foram interiorizados ao longo das aulas teóricas práticas e teóricas.

Com a realização deste trabalho, chegamos à conclusão que o protocolo de ligação de dados é fundamental em qualquer rede informática. A transferência de um ficheiro entre terminais diferentes deve garantir sempre a integridade de todos os dados transferidos. A realização do relatório foi também importante para conseguimos consolidar a matéria abordada ao longo do trabalho.

# Índice

[Sumário 2](#_Toc434612841)

[1. Introdução 4](#_Toc434612843)

[2. Arquitetura 4](#_Toc434612844)

[3. Estrutura do código 5](#_Toc434612845)

[4. Casos de uso principais 6](#_Toc434612846)

[5. Protocolo de ligação lógica 6](#_Toc434612847)

[6. Protocolo de aplicação 7](#_Toc434612848)

[7. Validação 7](#_Toc434612849)

[8. Elementos de valorização 8](#_Toc434612850)

[Seleção de parâmetros pelo utilizador 8](#_Toc434612851)

[Geração aleatória de erros em tramas de Informação 8](#_Toc434612852)

[Implementação de REJ 8](#_Toc434612853)

[Verificação da integridade dos dados pela Aplicação 8](#_Toc434612854)

[Registo de ocorrências 8](#_Toc434612855)

[9. Conclusões 8](#_Toc434612856)

[Anexo I 9](#_Toc434612857)

# 1. Introdução

O trabalho realizado ao longo das primeiras aulas práticas da unidade curricular de Redes de Computadores foram uma excelente introdução ao objetivo final deste 1º trabalho laboratorial. O objetivo era implementar um protocolo de ligação de dados. Nas primeiras aulas práticas, começamos pela transferência de simples caracteres, acabando agora a transferir uma imagem em formato *gif*, sendo que outros formatos e ficheiros seriam suportados pela nossa solução final, tendo em conta a sua robustez.

O guião, com todas as especificações exigidas, serviu de suporte à realização do trabalho. O protocolo de dados é baseado num conjunto de tramas, responsáveis por estabelecer e responder a todos os eventos da ligação. Ao longo deste relatório serão apresentadas todas as fases de implementação, caracterizando-as e apresentando as suas funcionalidades.

Todo o projeto foi desenvolvido em ambiente Linux, com utilização exclusiva da linguagem C e utilizando portas de série RS-232, com comunicação assíncrona.

# 2. Arquitetura

O programa é facilmente dividido em dois módulos, o emissor e o recetor. Apesar dessa divisão, o programa final é apenas um e cabe-lhe a ele interpretar, de acordo com o *input* do utilizador, qual o funcionamento adequado a cada momento.

Existem duas camadas, que permitem a correta funcionalidade do projeto, são elas a camada do protocolo de ligação de dados e a camada de aplicação. A camada de ligação de dados disponibiliza funções genéricas do protocolo. A camada de aplicação é responsável pela transferência de ficheiros, uma vez que aqui são executadas as funções responsáveis pela receção e emissão de tramas.

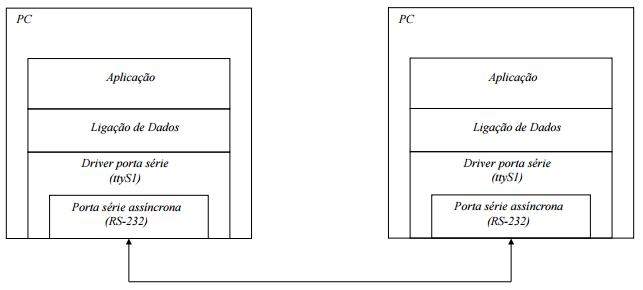


Figura 1 - Arquitetura

É importante também referir a existência de camadas responsáveis pela interação direta na porta de série, nomeadamente a camada do driver que permite o uso das funções da API do Linux para leitura e escrita de tramas.

A interface da aplicação, definida num ficheiro próprio, é responsável pelo estabelecimento de valores variáveis e apresentação da evolução da transferência. Os valores introduzidos são verificados, de modo a que sejam apenas admitidos valores válidos. A interface interage com a camada de aplicação, de modo a esta iniciar com os valores atribuídos pelo utilizador ou, em caso de erro na introdução, com os valores estabelecidos por defeito.

# 3. Estrutura do código

Nos ficheiros ApplicationLayer.c, ApplicationLayer.h e ApplicationLayerStruct.h encontra-se a implementação da camada de aplicação e as suas funções principais. Usamos uma *struct* para guardar os dados desta camada, em que o nome dos elementos são indicativos da sua funcionalidade.

typedef struct applicationLayerStruct {

char port[20]; //Dispositivo /dev/ttySx, x = 0, 1

int status; //TRANSMITTER | RECEIVER

int fdFile;

char fileName[128];

int fileSize;

int fileTransfer;

char currentPacket[MAX\_SIZE];

int currentPacketSize;

bool endFile;

int sequenceNumber;

int maxPacketSize;

} applicationLayer;

As principais funções da camada de aplicação são as seguintes:

int createInitialControlPacket(char \*packet, char \*fileName, int fileSize);

int createFinalControlPacket(char \*packet, char \*fileName, int fileSize);

void createDataPacket(applicationLayer \*applicationStruct);

bool readFromControlPacket(char \*packet, applicationLayer \*applicationStruct);

Os ficheiros LinkLayer.c, LinkLayer.h e LinkLayerStruct.c têm a implementação da camada de ligação de dados e as suas funções principais. Tal como na camada de aplicação, os dados estão guardados numa *struct*.

typedef struct linkLayerStruct {

int fd; //Descritor correspondente à porta de série

int baudRate; //Velocidade de transmissão

char sequenceNumber; //Número de sequência da trama: no formato (0 ou 1)

unsigned int timeout; //Valor do temporizador: 1 s

unsigned int numTransmissions; //Número de tentativas em caso de falha

char frame[FIELD\_MAX\_SIZE]; //Trama

int frameSize;

struct termios savetio;

} linkLayer;

As principais funções da camada de ligação de dados são as seguintes:

int llwrite(char \* buffer, int length, linkLayer \*linkStruct);

int llread(char \* buffer, linkLayer \*linkStruct);

void sendREJ(int fd, char sequenceNumber);

void sendRR(int fd, char sequenceNumber);

bool checkDisc(char \*frame, int frameSize);

bool checkSet(linkLayer \*linkStruct);

bool llopen(char \*portName, int option, linkLayer \*linkStruct);

int llclose(linkLayer \*linkStruct, int option);

Procurou-se dividir e estruturar o código, de forma que cada componente tivesse o seu próprio ficheiro e acabasse por ser mais fácil interpretar. Assim, existem outros ficheiros, para além dos apresentados anteriormente, são eles:

* Alarm.c e Alarm.h – tem a definição da rotina de alarme e a função para instalar o alarme.
* Cli.c e Cli.h – responsável pelas funções de interação com o utilizador.
* DefaultValues.h – ficheiro com a declaração de todos os valores por defeito.
* FieldProcessor.c e FieldProcessor.h – contêm funções auxiliares para a geração e interpretação das tramas.
* GlobalVariable.c e GlobalVariable.h – variáveis globais e respetivas funções de manipulação. Aqui estão os dados para a geração das estatísticas.
* main.c – programa principal.
* StateMachine.c e StateMachine.h – máquina de estados essencial para o funcionamento da aplicação.
* Transmission.c e Transmission.h – funções relacionadas com o estabelecimento da ligação da porta de série e o envio de sinais.

# 4. Casos de uso principais

A aplicação possui diversos casos de uso, sendo possível analisar esses casos de acordo com o tipo de utilização num dado momento. No entanto, existem funções que são utilizadas, independentemente de ser recetor ou emissor. Assim, a função **llopen** é usada para estabelecer a conexão. A função **llclose**, usada para fechar a conexão.

Caso o programa esteja a ser executado como emissor, usamos a função **openReadFile**, para abrir o ficheiro a ser transferido. Em seguida, cria-se a trama inicial através da função **createInitialControlPacket**, que é enviada, como todas as outras, através da função **llwrite.** Várias tramas de dados são criadas pela função **createDataPacket** e enviadas. Quando se chega ao final do ficheiro, cria-se uma trama final, através da função **createFinalControlPacket**. Essa trama é enviada com indicação para o programa terminar.

De outro modo, quando está a ser executado o recetor, as tramas são lidas através da função **llread.** O ficheiro de destino é aberto pela função **openWriteFile** e os dados são escritos nele através da função **writeFileFromDataPacket**.

Na iteração com o utilizador, realçamos a função **makeChoices** , utilizada para preencher a *struct*, com os valores definidos pelo utilizador e a função **startValues**, que atribui esses valores à *struct* da *Application Layer*.

# 5. Protocolo de ligação lógica

O envio e receção de tramas é feito através das funções implementadas ao nível da ligação de dados. É também nesta camada que estão implementadas as funções que constituem a API da porta de série.

A função **llopen** é responsável por estabelecer uma ligação através da porta de série. O alarme é inicializado e uma trama SET é enviada, esperando-se um comando UA em resposta pelo recetor, para o processo continuar. Se isto não acontecer, o processo é reiniciado com a ativação do alarme, um número definido de vezes, que se for atingido termina o programa com erro.

A função **llwrite** recebe um buffer que tenta escrever para a porta de série, ficando a aguardar a receção de uma resposta. Em caso de time-out, ou seja, a resposta não chegar no intervalo de tempo definido, é feita uma nova tentativa de envio da mensagem. Caso a resposta ser o comando RR, a mensagem foi transmitida corretamente. No entanto, se a resposta for o comando REJ, a mensagem não foi transmitida corretamente e, por isso, a mensagem é retransmitida.

A função **llread** recebe uma mensagem através da porta de série. Caso a mensagem seja inválida, procura-se fazer o tratamento e verificar o motivo. O comando REJ é enviado através da porta de série, quando se confirma que a mensagem é de facto inválida. Caso seja o comando DISC, que tenha sido recebido, significa que a ligação deve ser terminada. Se uma mensagem de informação for recebida, a informação é guarda e o comando RR enviado pela porta de série.

A função **llclose** termina a ligação através da porta de série. Recebe o comando DISC, indicando o fim da transmissão e emite um novo DISC para o emissor. Finalmente, o emissor envia um comando UA.

# 6. Protocolo de aplicação

A camada de aplicação é a camada de mais alto nível, responsável pelos pacotes de controlo, pacotes de dados e transmissão do ficheiro. A camada de aplicação consegue diferenciar estes dois tipos de pacotes através do primeiro byte do pacote, denominado campo de controlo.

Os pacotes de controlo marcam o início e o fim da transmissão de um ficheiro.

int createInitialControlPacket(char \*packet, char \*fileName, int fileSize);

int createFinalControlPacket(char \*packet, char \*fileName, int fileSize);

Os pacotes de dados são os pacotes que transportam as tramas de informação.

void createDataPacket(applicationLayer \*applicationStruct);

O envio e receção do ficheiro é tratado na máquina de estados, presente no ficheiro main.c. Inicialmente, envia-se o pacote de controlo inicial com o tamanho e o nome do ficheiro a ser transmitido. Seguidamente são enviados os pacotes de dados, repetidamente, até a totalidade do ficheiro ser enviada.

# 7. Validação

Durante a realização do trabalho, procurou-se testar ao máximo todos os cenários possíveis. Para além da imagem usada no teste com o docente, testamos a transferência de outras imagens e até ficheiros áudio. Sendo que todas essas transferências correram dentro da normalidade.

Além da transferência normal, testamos também a extração do cabo da porta de série, como era objetivo do trabalho. Todos estes testes foram superados, como se comprovou na avaliação.

# 8. Elementos de valorização

## Seleção de parâmetros pelo utilizador

Na interface inicial, é possível selecionar a *baud rate*, o tamanho máximo do campo de Informação das tramas I, o número máximo de retransmissões e o intervalo de time-out.

## Geração aleatória de erros em tramas de Informação

Através de uma variável, que define a probabilidade de erro por cada trama I, é possível simular a ocorrência de erro no cabeçalho e no campo de dados. Esses erros são tratados tal como um erro real. Essa probabilidade de erro é usada na seguinte condição:

if (stateMachineRead(&sizeRet, buffer, linkStruct) && !((rand() % ERROR\_MAX) < ERROR\_ODD))

## Implementação de REJ

No caso de ocorrer um erro do tipo BCC2 na função llread, o comando REJ é enviado para que o emissor reenvie a mensagem que não chegou ao recetor corretamente. A implementação deste mecanismo está na função:

void processDamagedDataI(linkLayer \*linkStruct)

## Verificação da integridade dos dados pela Aplicação

A aplicação verifica o tamanho do ficheiro recebido, certificando-se que é igual ao enviado. A numeração de cada pacote garante que pacotes duplicados são ignorados. Verificamos também que nenhum pacote é perdido, caso isso aconteça, aparece uma mensagem de erro ao utilizador, reportando a situação.

## Registo de ocorrências

Ao longo da execução da aplicação, vão sendo registados as várias ocorrências de RR, REJ, bem como o número de mensagens enviadas e recebidas. Esses dados vão sendo atualizados nas variáveis globais respetivas.

# 9. Conclusões

Ao longo da realização deste trabalho, foi muito importante analisar minuciosamente todos os critérios especificados no guião. No final, conseguiu-se um programa robusto, capaz de realizar transferências de dados entre computadores, cumprindo a totalidade dos requisitos apresentados.

Com a realização deste trabalho, verificamos a real importância dos protocolos de ligações de dados, assim como a quantidade de erros que podem existir e devem ser tratados. A simples transferência de um ficheiro é na verdade um processo complexo e deveras exigente.

A realização deste projeto contribuiu para a consolidação dos conceitos interiorizados nas aulas teóricas, para um conhecimento mais profundo da comunicação em redes de computadores e para trabalhar diretamente com a porta de série.

# Anexo I

Código fonte

(SerialPort.zip)