

**Redes de Computadores**

**Mestrado Integrado de Engenharia Informática e Computação, 3ºano**

10 de novembro de 2020

Protocolo de Ligação de Dados

Daniel Garcia Silva, up201806524@fe.up.pt

Mariana Truta, up201806543@fe.up.pt

Turma 3

Índice

[1. Sumário 2](#_Toc55840633)

[2. Introdução 2](#_Toc55840634)

[3. Arquitetura 3](#_Toc55840635)

[4. Estrutura de código 3](#_Toc55840636)

[appSender 3](#_Toc55840637)

[appReceiver 3](#_Toc55840638)

[dataLink 4](#_Toc55840639)

[port 4](#_Toc55840640)

[protocol 4](#_Toc55840641)

[stateMachines 5](#_Toc55840642)

[macros 5](#_Toc55840643)

[5. Casos de uso principais 6](#_Toc55840644)

[6. Protocolo de ligação lógica 6](#_Toc55840645)

[7. Protocolo de aplicação 6](#_Toc55840646)

[8. Validação 6](#_Toc55840647)

[9. Eficiência do protocolo de ligação de dados 6](#_Toc55840648)

[10. Conclusão 7](#_Toc55840649)

# Sumário

No âmbito da unidade curricular de *Redes de Computadores*, foi elaborado um projeto que consistia no desenvolvimento de um *software* que permitisse a **transferência** de ficheiros de um computador para o outro, estando estes ligados por um **cabo série**.

Ao longo deste relatório, será explicado como foram **cumpridos** todos os objetivos do projeto, tendo sido concluída uma aplicação funcional e sem perdas de dados.

# Introdução

Este primeiro projeto tinha dois grandes objetivos: implementar um **protocolo de ligação de dados**, especificado no guião fornecido pelos docentes, e testá-lo com uma **aplicação** simples de transferência de ficheiros. Relativamente ao ambiente de desenvolvimento, o trabalho foi realizado em LINUX, utilizando a linguagem de programação C e portas série RS-232 cuja comunicação é assíncrona.

Neste relatório, pretende-se tornar claro como foi possível a elaboração de um **serviço de comunicação fiável** entre dois computadores por via de uma porta de série assíncrona, apresentando detalhes de toda a **teoria** utilizada. Este relatório está estruturado da seguinte forma:

* **Arquitetura:** descriçãodosblocos funcionais e interfaces implementadas;
* **Estrutura de código:** apresentação das *APIs*, principais estruturas de dados e funções e sua relação com a arquitetura;
* **Casos de uso principais:** identificação dos casos de uso e sequências de chamadas de funções;
* **Protocolo de ligação de dados:** identificação dos principais aspetos funcionais bem como a descrição da estratégia de implementação dos mesmos, sendo complementada com a apresentação de extratos de código;
* **Protocolo de aplicação:** identificação dos principais aspetos funcionais bem como a descrição da estratégia de implementação dos mesmos, sendo complementada com a apresentação de extratos de código;
* **Validação:** descrição dos testes efetuados com apresentação quantificada dos resultados;
* **Eficiência de protocolo de ligação de dados:** caraterização estatística da eficiência do protocolo, recorrendo a medidas sobre o código desenvolvido;
* **Conclusão:** síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados.

# Arquitetura

O projeto está dividido em duas camadas muito bem definidas: a camada de **protocolo de ligação de dados** e a camada da **aplicação**.

O objetivo do **protocolo de ligação de dados** é fornecer um serviço de comunicação de dados **fiável** entre dois sistemas ligados por um **cabo série**. Esta camada contem, assim, todas as funções necessárias para a **abertura**, **fecho**, **escrita** e **leitura** da porta de série. Para além disso, tem também a seu encargo o controlo de **erros** e **fluxo** e o *stuffing*/*destuffing* de pacotes.

A camada da **aplicação** está situada acima da camada de ligação de dados, sendo responsável pelo **envio** e **receção** de ficheiros. Tem de permitir enviar e identificar os **pacotes de controlo** bem como **dividir** o ficheiro em vários pacotes, no caso do emissor, ou **concatenar** e **interpretar** toda a informação recebida, no caso do recetor.

É importante referir que esta arquitetura baseia-se no **principio de independência entre camadas**, isto é, ao nível da camada de ligação de dados, não existe qualquer distinção ou processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de informação e a camada da aplicação não conhece os detalhes da outra camada mas apenas a forma como acede ao serviço.

# Estrutura de código

A **camada da** **aplicação** está dividida em dois ficheiros: *appSender*, que é a parte utilizada pelo emissor, e *appReceiver*, que é utilizada pelo recetor. As funções utilizadas são as seguintes:

## appSender

* ***main*** -é responsável pela escrita de um ficheiro, sendo o *path* fornecido pelo utilizador.

## appReceiver

* ***main*** – é responsável pela litura de um ficheiro, sendo guardado em *“./images/ToReceive”*;
* ***freeFile*** - liberta a memória que foi alocada à *struct* *File;*
* ***parseInfo*** - Encaminha o pacote de dados para função respetiva, tendo em conta

o seu primeiro byte (*START\_BYTE*, *DATA\_BYTE* e *END\_BYTE*);

* ***parseControlPacket*** - Interpreta a informação guardada na *info*, armazenando o nome e o tamanho do ficheiro que irá ser recebido na *struct File*;
* ***checkControlPacket*** - Verifica se o pacote de controlo de finalização contém a mesma informação que o pacote de controlo que sinalizou o inicio da transmissão, diferenciando apenas no campo de controlo;
* ***parseDataPacket*** - guarda os dados do ficheiro no campo *data* da *struct File*;
* ***initFile*** - inicializa a *struct File*, alocando a memória necessária.

## dataLink

Por sua vez, a **camada de ligação de dados** está dividida em quatro principais funções que recorrem a ficheiros e funções auxiliares:

* ***llopen*** - estabelece a ligação entre o emissor e o recetor, usando um mecanismo de *Stop and Wait* e garantindo que se reúnem as condições necessárias para começar a transmissão de dados;
* ***llwrite*** - é responsável pela incorporação, numa trama *I*, de um conjunto de *bytes*, *buffer,* sendo realizado o *byte stuffing* do pacote, e pelo envio desta mesma trama para o recetor. Para além disso, tem de ser capaz de agir de acordo com uma mensagem *ACK* que o recetor irá enviar como resposta;
* ***llread***- é responsável pela leitura de uma mensagem enviada pelo recetor e após o *byte destuffing* do pacote, envia uma resposta de rejeição, no caso do pacote não ser válido, ou de aceitação, se o pacote for válido;
* ***llclose*** - termina a ligação estabelecida anteriormente entre o emissor e o recetor e envia e recebe tramas de supervisão para indicar que terminou com sucesso.

Para tornar o código mais limpo e organizado, foram criadas alguns ficheiros e funções auxiliares:

## port

* ***initPort*** - abre a porta série, guarda os seus atributos na respetiva *struct termios* e retorna o descritor da porta;
* ***closePort*** - fecha a porta série.

## protocol

* ***alarmSenderHandler***/***alarmReceiverHandler*** – são chamadas quando o tempo do alarme do emissor ou do recetor, respetivamente, se esgota;
* ***SandWOpenClose*** - envia uma mensagem com o comando *send* e o endereço *sendAddress* e espera por uma resposta com o comando *receive* e o endereço *recAddress*, usando um mecanismo de *Stop and Wait;*
* ***sendOpenCloseFrame*** – cria e envia uma trama de acordo com o *comando* (*SET, DISC* ou *UA*) e o *endereço* passados como argumentos;
* ***sendAckFrame*** - cria e envia uma trama de rejeição ou de aceitação;
* ***receiveOpenCloseFrame*** - lê a mensagem recebida (*SET*, *DISC* ou *UA*), chamando a função *changeStateS* que irá interpretar o byte e mudar de estado de acordo com o mesmo;
* ***receiveAckFrame*** - lê uma trama ACK, chama a função que verifica se é válida ou não, *changeStateAck*, e retorna 1, caso tenha sido rejeitada, ou 0, se for válida;
* ***sendInfoFrame*** - cria e envia uma trama *I* com os dados em *info*, acrescentando a quantidade necessária de *0x00* até ser atingido o valor *IFRAME\_SIZE – 2* e garantindo assim que a *frame* tem sempre um tamanho fixo;
* ***receiveInfoFrame*** - é responsável por ler uma trama *I*, guardando o pacote de controlo ou de dados em *info*, após o *byte destuffing.* O seu valor de retorno depende se a trama foi ignorada, rejeitada ou aceite ou se o tempo do alarme esgotou;
* ***makeControlPacket*** – cria um pacote de controlo com o campo de controlo *control*, com o tamanho do ficheiro *fileSize* e com o nome do ficheiro *fileName*.;
* ***makeDataPacket*** - cria um pacote de dados com o número de sequência *N* e os dados *info*.

## stateMachines

* ***changeStateS*** - verifica se o byte recebido é o esperado e age de acordo com o mesmo, atualizando o estado atual da trama *SET*, *DISC* ou *UA*;
* ***changeStateInfo*** - verifica se o byte recebido é o esperado e age de acordo com o mesmo, atualizando o estado atual da trama *I*;
* ***changeStateAck*** - verifica se o byte recebido é o esperado e age de acordo com o mesmo, atualizando o estado atual da trama *REJ ou RR*.

## macros

Neste ficheiro, estão guardadas todas as macros necessárias ao longo do projeto, devidamente comentadas.

***dataStructures***

Neste ficheiro, estão guardadas as principais estruturas de dados utilizadas, que permitem guardar o estado atual da respetiva trama (*State, AckState*), guardar o tipo de comando de controlo (*Command*) e guardar a informação do ficheiro recebido (*File*).

Em anexo, encontra-se um pequeno esquema dos ficheiros utilizados.

# Casos de uso principais

Pode-se identificar como principais casos a **interface,** que permite escolher a porta de série a usar e o ficheiro a enviar, e a **transferência** do ficheiro pela porta série.

Inicialmente, é necessário compilar o programa, executando o comando *make* na pasta *src.* De seguida, é necessário executar *“./sender <porta> <path do ficheiro a enviar>”*, no caso do emissor, e *“./receiver <porta>”,* no caso do recetor. A porta de série tem de estar no formato *“/dev/ttySX”* sendo X o número da mesma.

A **transmissão**/**receção** dos dados é feita pela seguinte ordem:

|  |  |
| --- | --- |
| no caso do **emissor**,   1. Estabelecimento da ligação entre o emissor e recetor na função *llopen*; 2. Criação do pacote de controlo *START* e o seu envio com *llwrite*; 3. Criação de pacotes de dados com os dados do ficheiro e o seu envio em *llwrite*; 4. Criação do pacote de controlo *END* e o seu envio em *llwrite*; 5. Terminação da ligação entre o emissor e o recetor na função *llclose.* | no caso do **recetor**,   1. Estabelecimento da ligação entre o emissor e recetor na função *llopen*; 2. Receção e análise do pacote de controlo *START*, enviando uma resposta; 3. Receção e análise dos pacotes de dados, enviando uma resposta; 4. Receção e análise do pacote de controlo *END*; 5. Escrita dos dados recebidos num ficheiro com o mesmo nome do ficheiro enviado pelo emissor, sendo guardado na pasta *imagesToReceive;* 6. Terminação da ligação entre o emissor e o recetor na função *llclose.* |

# Protocolo de ligação lógica

# Protocolo de aplicação

# Validação

# Eficiência do protocolo de ligação de dados

# Conclusão