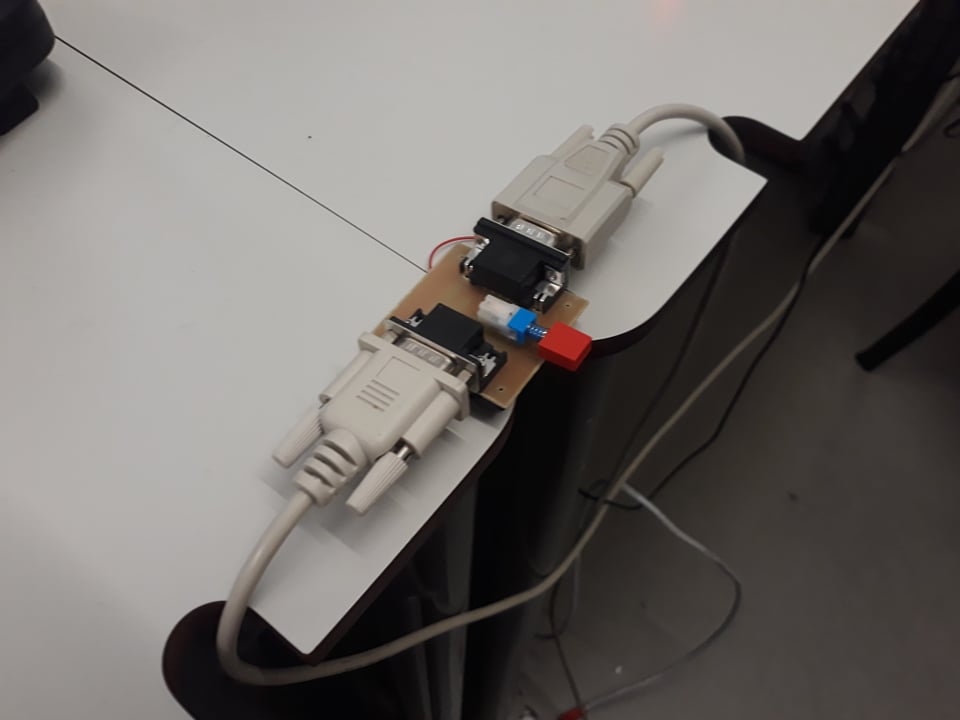


MIEIC 3º Ano

1º Trabalho Laboratorial

Ligação de Dados



**Trabalho realizado por:**

Diogo Machado up201706832@fe.up.pt  
Eduardo Ribeiro up201705421@fe.up.pt  
José Guerra up201706421@fe.up.pt

**Unidade Curricular:** RCOM

**Regente:** Manuel Alberto Pereira Ricardo

**Ano Letivo:** 2019/2020

**Data de Entrega**: 28 de outubro de 2019

**Sumário**

Este relatório foi realizado no âmbito da unidade curricular redes de computadores (RCOM) do 3º ano do mestrado integrado em engenharia informática e de computação (MIEIC). O relatório incide sobre o primeiro trabalho laboratorial realizado cujo foco é a transferência de dados através de uma aplicação. A transferência de dados é feita através da implementação de um protocolo de comunicação entre duas máquinas. O trabalho prático foi concluído com sucesso sendo todos os objetivos definidos no início alcançados.

O relatório serve para detalhar a implementação do trabalho bem como explicar os conceitos teóricos que foram aplicados.

**Introdução**

O objetivo deste trabalho pode subdividir em dois objetivos concretos, o objetivo relativo a camada do protocolo de ligação de dados e o objetivo relativo a camada da aplicação. O principal objetivo do protocolo de Ligação de Dados é fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio (canal) de transmissão – neste caso, um cabo série. O objetivo da aplicação é desenvolver um protocolo de aplicação muito simples para transferência de um ficheiro, usando o serviço fiável oferecido pelo protocolo de ligação de dados.

O objetivo do relatório é explicar ao leitor a parte teórica deste trabalho, bem como a nossa implementação do que foi proposto pelo guião, tendo a seguinte estrutura:

* Arquitetura - Visualização dos blocos funcionais e interfaces.
* Estrutura do código - Representação das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura.
* Casos de uso principais - Identificação dos mesmos bem com as sequências de chamada de funções.
* Protocolo de ligação lógica - Identificação dos principais aspetos funcionais bem como a descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.
* Protocolo de aplicação - Identificação dos principais aspetos funcionais bem como a descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.
* Validação - Descrição dos testes efetuados com apresentação quantificada dos resultados.
* Eficiência do protocolo de ligação de dados - Caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.
* Conclusões - Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

**Arquitetura**

O trabalho está dividido em dois blocos: o writer (emissor) e o reader (leitor). Ambos os blocos chamam funções presentes na camada de aplicação e na camada de ligação de dados porém há uma clara distinção de quem está a utilizar as estas camadas, logo embora ambos utilizem o llopen() e llclose() (funções da aplicação de dados) a implementação destas funções garante que haja independência entre emissor e leitor.

**Estrutura de código**

O código encontra-se dividido em dois ficheiros principais (mains): o WritenonCanonical.c que corresponde ao writer; o noncanonical.c que corresponde ao reader. Ambos estes ficheiros chamam funções da aplicação e esta por sua vez chama funções da ligação de dados.

**writenoncanonical (writer)**

Funções principais da camada de ligação

* **llopen()** - envia trama de supervisão *SET* e recebe trama *UA*
* **llwrite()** - efetua *byte stuffing* das *I-frames* e envia-as para o recetor
* **llclose()** - envia trama de supervisão *DISC*, recebe trama *DISC* e envia trama *UA*

Funções principais da camada de aplicação

* **sendFile()** - abre o ficheiro, separa-o em pacotes, envia-os dentro de tramas para o recetor e fecha o ficheiro
* **openFile()** - efetua a chamada ao sistema que abre o ficheiro a enviar
* **closeFile()** - fecha o ficheiro após o seu envio
* **getFileSize()** - obtém o tamanho de um ficheiro aberto
* **buildControlPacket()** - constrói um pacote de controlo, para ser enviado numa *I-frame*
* **buildDataPacket()** - constrói um pacote de dados do ficheiro, para ser enviado numa *I-frame*

Estruturas de dados

**struct linkLayer** {  
 char port[20]; /\* Dispositivo /dev/ttySx, x = 0, 1 \*/  
 int baudRate; /\* Velocidade de transmissão \*/  
 unsigned int sequenceNumber; /\* Número de sequência da trama: 0, 1 \*/  
 unsigned int timeout; /\* Valor do temporizador: 1 s \*/  
 unsigned int numTransmissions; /\* Número de tentativas em caso de falha \*/  
 unsigned char frame[MAX\_SIZE\_FRAME]; /\* Trama \*/  
 unsigned int frameLength; /\* Comprimento atual da trama \*/  
 };

**struct applicationLayer** {  
 int fileDescriptor; /*\** Descritor correspondente à porta série *\**/  
 int status; /*\** TRANSMITTER | RECEIVER *\**/  
 };

Variáveis globais

* **struct linkLayer ll;**
* **struct applicationLayer al;**
* **struct termios oldtio;**

Macros pertinentes:

* **TRANSMITTER**
* **MAX\_PACK\_SIZE**
* **MAX\_DATA\_SIZE**
* **CTRL\_START**
* **CTRL\_DATA**
* **CTRL\_END**

**noncanonical (reader)**

Funções principais da camada de ligação

* **llopen()** - recebe trama de supervisão *SET* e envia trama *UA*
* **llread()** - recebe as *I-frames*, lê-as, e efetua *byte destuffing*
* **llclose()** - recebe trama de supervisão *DISC*, envia trama *DISC*, e recebe trama *UA*

Funções principais da camada de aplicação

* **receiveFile()** - recebe o ficheiro, recebe as tramas e guarda-os pacotes no novo ficheiro
* **openFile()** - efetua a chamada ao sistema que cria o ficheiro onde vai ser guardada a informação recebida
* **closeFile()** - fecha o novo ficheiro após recebê-lo na totalidade
* **getFileSize()** - obtém o tamanho de um ficheiro aberto
* **parseControlPacket()** - extrai a informação de um pacote de controlo de uma *I-frame*
* **parseDataPacket()** - extrai a informação de um pacote de dados do ficheiro de uma *I-frame*

Estruturas de dados (igual a **writenoncanonical (writer)**, acima)

**struct linkLayer** {  
 char port[20]; /\* Dispositivo /dev/ttySx, x = 0, 1 \*/  
 int baudRate; /\* Velocidade de transmissão \*/  
 unsigned int sequenceNumber; /\* Número de sequência da trama: 0, 1 \*/  
 unsigned int timeout; /\* Valor do temporizador: 1 s \*/  
 unsigned int numTransmissions; /\* Número de tentativas em caso de falha \*/  
 unsigned char frame[MAX\_SIZE\_FRAME]; /\* Trama \*/  
 unsigned int frameLength; /\* Comprimento atual da trama \*/  
 };

**struct applicationLayer** {  
 int fileDescriptor; /*\** Descritor correspondente à porta série *\**/  
 int status; /*\** TRANSMITTER | RECEIVER *\**/  
 };

Variáveis globais

* **struct linkLayer ll;**
* **struct applicationLayer al;**
* **struct termios oldtio;**

Macros pertinentes

* **RECEIVER**
* **MAX\_PACK\_SIZE**
* **MAX\_DATA\_SIZE**
* **CTRL\_START**
* **CTRL\_DATA**
* **CTRL\_END**

**Casos de Uso Principais**

**Interface**

A interface permite ao transmissor escolher o ficheiro a enviar.  
O utilizador, utilizando a consola, correrá o programa, dando um conjunto de argumentos. Do lado do emissor, deve ser inserida a porta de série a ser utilizada (ex. **/dev/ttyS0**) e o nome do ficheiro a ser enviado (ex. **pinguim.gif**). Do lado do recetor, apenas deve ser inserida a porta de série.

**Transmissão de Dados**

A transmissão de dados ocorre via porta de série, entre dois computadores, dando-se a seguinte sequência de eventos:

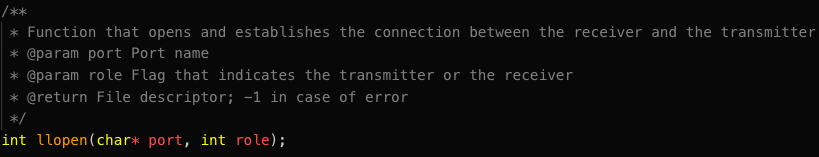
* O utilizador, do lado do transmissor, escolhe o ficheiro a ser enviado;
* É configurada a ligação entre os dois computadores;
* A ligação é estabelecida;
* O transmissor envia os dados, trama a trama;
* Simultaneamente, o recetor recebe os dados, trama a trama;
* À medida que recebe os dados, o recetor guarda-os num ficheiro com o mesmo nome do ficheiro enviado pelo emissor;
* A ligação é terminada.

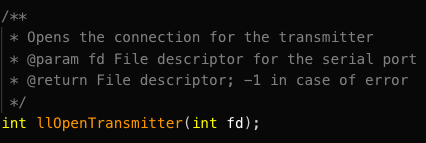
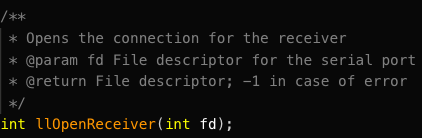
**Protocolo de ligação lógica**

**Identificação dos principais aspetos funcionais**

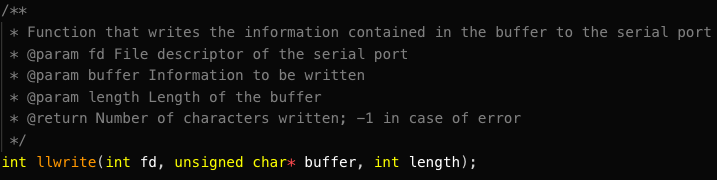
**Protocolo de ligação de dados**

No protocolo de ligação lógica, tal como é pedido no guião do trabalho, foram implementadas as funções **llopen, llwrite, llread** e **llclose**. São as principais funções deste protocolo, e servem de interface para o protocolo de aplicação usar as funcionalidades do data link. Estas funções, respetivamente, fazem: o estabelecimento da ligação entre o emissor e o recetor; o envio de uma trama de informação, até este ter sucesso; a receção de uma trama de informação, até este ter sucesso; e a finalização da ligação entre o emissor e o recetor.

**LLOPEN**

Esta função começa por preencher os campos da estrutura *LinkLayer* com os valores corretos, abrir a ligação à porta de série e criar o descritor de ficheiro, e por instalar o alarm handler de modo a poder dar os timeouts. De seguida, tendo em conta o valor do parâmetro **role**, chama uma das duas funções auxiliares, **llOpenTransmitter** e **llOpenReceiver**.

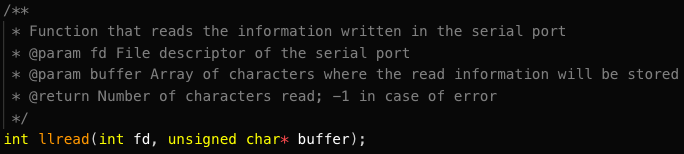
Estas funções desempenham o resto das funcionalidades que o emissor e recetor deveriam ter no llopen. O emissor envia um trama de supervisão SET e fica a espera de ler um trama UA, ocorrendo o timeout e/ou a saída do programa se o número máximo de retransmissões for ultrapassado. O recetor lê um trama SET, e depois de o receber envia um trama UA. O envio de tramas é feito com a função **sendFrame**, e receção com **readSupervisionFrame**. As tramas são geradas através da função **createSupervisionFrame**. A receção das tramas é feita byte a byte, e o valor de cada um é processado através de uma máquina de estados, através da função **changeState**. O envio é feito trama a trama.

**LLWRITE**

Esta função é apenas chamada pelo emissor, de modo a enviar tramas de informação ao recetor, e é composta por várias fases:

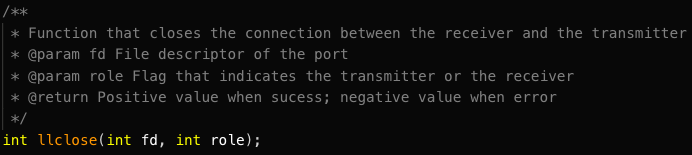
* + - * + Criação da trama de informação, utilizando os dados da aplicação passados como argumento (**createInformationFrame**);
        + Introdução de *byte stuffing* na trama gerada (**byteStuffing**);
        + Envio da trama, com o número de sequência correto (S\_0 ou S\_1) (**sendFrame**);
        + Leitura da resposta (RR ou REJ), com possibilidade de timeout (**readSupervisionFrame**);
        + Saída da função, se foi recebido um RR, ou reenvio da trama, se for recebido um REJ, as vezes que forem necessárias até receber uma confirmação positiva.

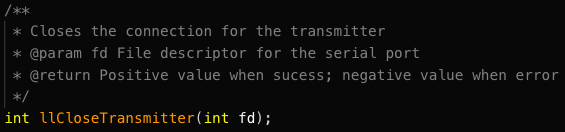
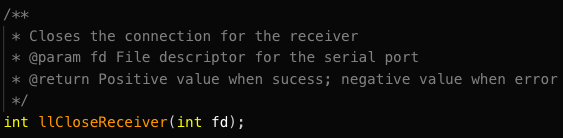
O timeout das tramas I, SET e DISC é feito da seguinte maneira. **finish** e **resendFrame** são flags modificadas pelo alarm handler, quando ocorre um timeout, que indicam se o programa deve acabar ou se a trama deve ser reenviada, respetivamente.

**LLREAD**

Esta função é apenas chamada pelo recetor, de modo a receber tramas de informação provenientes do emissor, e é composta por várias fases:

* + - * + Leitura da trama de informação enviada pelo emissor (**readInformationFrame**);
        + Utilização de *byte destuffing* na trama recebida (**byteDestuffing**);
        + Verificação do BCC2 (**createBCC2**, e comparação com o BCC2 recebido na trama);
        + Determinação da resposta que deve ser dada ao emissor, tendo em conta o resultado dos passos anteriores;
        + Criação e envio da trama de supervisão correta ao emissor (RR ou REJ, com o número de sequência de trama correto) (**createSupervisionFrame** e **sendFrame**);
        + Saída da função, se tudo funcionou corretamente e o buffer passado como argumento foi preenchido, ou tentativa de nova leitura de uma trama de informação, se algum erro ocorreu.

**LLCLOSE**

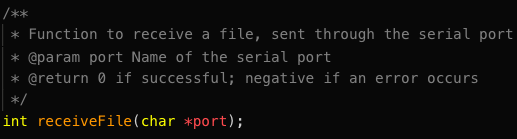
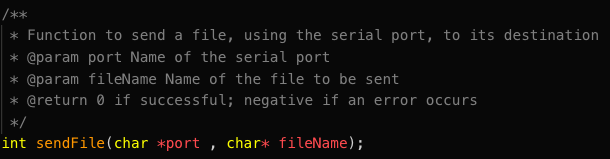
Tal como o llopen, esta função recebe um parâmetro **role**, que indica se é o emissor ou recetor a chamar a função.

O emissor envia um DISC, recebe um DISC do recetor e envia um UA, fechando a ligação através da porta de série. O recetor recebe o DISC do emissor, envia um DISC, e recebe um UA, fechando também a ligação posteriormente se não houver erros. Foi implementada a possibilidade de ocorrência de timeouts nas transmissões corretas.

**FUNÇÕES AUXILIARES**

Para além das funções descritas acima, várias outras funções auxiliares foram implementadas, que podem ser observadas melhor nos anexos.

**Protocolo de aplicação**

O protocolo da aplicação tem como funções principais as funções **sendFile** e **receiveFile**, que são chamadas nas funções main do emissor e do recetor, respetivamente. Estas funções fazem todas as tarefas que cada máquina deve executar.

**SENDFILE**

São a seguir listadas as principais funcionalidades da função sendFile:

* + - * + Abertura do ficheiro a ser enviado;
        + Estabelecimento da ligação com o recetor, utilizando a função **llopen**;
        + Envio do pacote de controlo START, usando **llwrite** (contém o nome e tamanho do ficheiro a enviar);
        + Fragmentação do ficheiro em pequenos pacotes, cujo tamanho é dependente do tamanho máximo especificado para os pacotes de dados, e envio de cada pacote, por ordem, ao recetor (monitorizando a ordem como o número de sequência), utilizando **llwrite**;
        + Envio do pacote de controlo END, usando **llwrite**;
        + Finalização da ligação com o recetor, usando **llclose**.

\

**RECEIVEFILE**

São a seguir listadas as principais funcionalidades da função receiveFile:

* + - * + Estabelecimento da ligação com o emissor, utilizando a função **llopen**;
        + Receção do pacote de controlo START, usando **llread** (contém o nome e tamanho do ficheiro a enviar);
        + Abertura de um novo ficheiro com o nome idêntico ao que foi passado no pacote de controlo START;
        + Receção, pacote a pacote, com **llread**, dos fragmentos de ficheiro enviados sequencialmente pelo emissor (sendo que a ordem é confirmada através do número de sequência). Cada fragmento é escrito no ficheiro aberto, de modo a juntar os fragmentos recebidos para recriar o ficheiro;
        + Fragmentação do ficheiro em pequenos pacotes, cujo tamanho é dependente do tamanho máximo especificado para os pacotes de dados, e envio de cada pacote, por ordem, ao recetor (monitorizando a ordem como o número de sequência), utilizando **llwrite**;
        + Receção do pacote de controlo END, usando **llread**;
        + Comparação do tamanho do ficheiro recriado com o tamanho de ficheiro passado nos pacotes de controlo;
        + Comparação das informações passadas nos pacotes de controlo START e END, para detetar quaisquer possíveis erros no envio por parte do emissor;
        + Finalização da ligação com o emissor, usando **llclose**.

**FUNÇÕES AUXILIARES**

Para além das funções descritas acima, foram feitas algumas funções auxiliares, mais precisamente para a construção dos pacotes de controlo e de dados (**buildDataPacket** e **buildControlPacket**), e para o parse destes pacotes, de modo a retirar as informações dos mesmos (**parseControlPacket** e **parseDataPacket**). Foram também feitas funções auxiliares de manipulação de ficheiros (**getFileSize**, **openFile** e **closeFile**).

**Informações Gerais**

Para todas as funções descritas, quer na camada de ligação de dados, quer na camada da aplicação, é verificado o valor de retorno, ocorrendo o tratamento de possíveis erros.

**Validação**

**Testes efetuados**

De forma a verificar o comportamento do programa, os seguintes testes foram efetuados:

* Envio de vários ficheiros, com diferentes tamanhos
* Interrupção da ligação por cabo entre as portas de série
* Geração de ruído na ligação entre as portas de série
* Envio de ficheiros com diferentes taxas de simulação de erros
* Envio de ficheiros com diferentes valores de *baudrate* (capacidade de ligação)
* Envio de ficheiros com diferentes tamanhos para as *I-frames*

**Resultados obtidos**

Todos os testes que foram efetuados foram concluídos com sucesso, verificando-se o comportamento que era esperado.

**Eficiência do protocolo de ligação de dados**

De modo a avaliar a eficiência do programa, foram efetuados os testes em seguida explicitados.

Cada valor obtido vem da média de 7 ensaios para as condições em questão, sendo eliminados os dois valores mais extremos (o menor e o maior), obtendo-se assim uma média de 5 valores, para minimizar o efeito de anomalias estatísticas.

Os testes foram executados com um ficheiro de imagem com 130KB, de modo a que o grande número de tramas tornasse o efeito das variações testadas homogéneo em todos os ensaios.

Excetuando o teste à variação desse mesmo valor, o tamanho máximo de pacote de informação utilizado foi de 1024B.

**Variação do FER**

Com o gráfico do valor de S em função da percentagem de erros simulados, podemos concluir que o FER tem um impacto significativo na eficiência do programa. Isto deve-se, primariamente, ao facto de que, quando é gerado um erro no *BCC1*, irá ocorrer um *timeout*, que resultará na ausência de resposta por parte do recetor, por um número previamente definido de segundos (no nosso caso, **3 segundos**), o que afeta negativamente o tempo de execução. Já os erros no *BCC2* não têm um impacto tão grande, pois apenas causam o reenvio da trama, que é imediato.

[gráfico]

**Variação do *baudrate* (C – capacidade de ligação)**

Com este gráfico, podemos concluir que, com o aumento da capacidade de ligação, diminui a eficiência.

[gráfico]

**Variação do tamanho das *I-frames***

Com o gráfico correspondente à variação de S, com base no tamanho das *I-frames*, mostra-nos que, quanto maior o tamanho de cada pacote de dados, mais eficiente será o programa. Isto verifica-se porque cada envio contém mais informação, o que reduz o número de tramas, pelo que o *timeout* irá ocorrer menos vezes.

[gráfico]

**Variação do tempo de propagação (T\_prop)**

**[texto]**

**[gráfico]**

**Conclusão**

**Síntese**

Este projeto consiste no desenvolvimento de um serviço fiável de comunicação entre dois computadores ligados por porta de série, implementando o **protocolo de ligação de dados**.

**Reflexão**

Este trabalho permitiu-nos compreender o protocolo de ligação de dados, incidindo sobre a estrutura das tramas e o processo de encapsulamento, envio e receção da informação.

Adicionalmente, é destacada a importância da **independência entre camadas**, sendo que cada camada do programa respeita esse conceito. A camada da ligação de dados não recorre, de todo, a qualquer processamento desenvolvido na camada da aplicação. Já a camada da aplicação não conhece quaisquer detalhes da implementação da camada da ligação de dados, conhecendo apenas como utilizar as suas funcionalidades.