1º Trabalho Laboratorial

Relatório



Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Beatriz de Henriques Martins – up201502858

Nádia de Sousa Varela Carvalho – up201208223

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

31 Outubro 2017

Índice

[1 Sumário 3](#_Toc497487637)

[2 Introdução 3](#_Toc497487638)

[3 Arquitetura e Estrutura do código 3](#_Toc497487639)

[3.1 Data Link – Ligação de dados 3](#_Toc497487640)

[3.2 App Link – Aplicação 4](#_Toc497487641)

[4 Casos de uso principais 4](#_Toc497487642)

[5 Protocolo de ligação lógica 4](#_Toc497487643)

[5.1 llopen() e llclose() 5](#_Toc497487644)

[5.2 llread() e llwrite() 5](#_Toc497487645)

[6 Protocolo de aplicação 5](#_Toc497487646)

[6.1 send\_Data() e receive\_Data() 5](#_Toc497487647)

[7 Validação 5](#_Toc497487648)

[8 Eficiência do protocolo de ligação de dados 5](#_Toc497487649)

[9 Conclusões 5](#_Toc497487650)

[Anexo – Código fonte 6](#_Toc497487651)

# Sumário

Este trabalho foi realizado no âmbito da cadeira *Redes de Computadores* com o objetivo de aplicar todos os conhecimentos lecionados nas aulas teóricas sobre o *Protocolo de Ligação de Dado*. O trabalho em si consiste na transferência de dados através de uma porta de série.

O projeto foi concluído. No entanto, não está completo, uma vez que a aplicação só é capaz de enviar o ficheiro quando não ocorrem erros.

# Introdução

O objetivo principal deste trabalho é a implementação de um determinado protocolo para transferência de dados, especificado no guião do projeto, através de uma porta de série, uma das formas mais básicas para transferência de dados.

O objetivo do relatório é demonstrar como a parte teórica foi aplicada ao projeto, uma vez que na apresentação/demonstração deste não foi muito visível.

A organização do relatório será a seguinte:

* **Introdução** – breve indicação dos objetivos do projeto e do relatório;
* **Arquitetura e Estrutura do código** – explicação das API’s, principais estruturas de dados, funções e a sua relação com a arquitetura;
* **Casos principais de uso** –identificação dos principais casos de uso e sequências de chamadas
* **Protocolo de ligação de lógica** –identificação dos principais aspetos funcionais e descrição da estratégia de implementação
* **Protocolo de aplicação** –identificação dos principais aspetos funcionais e descrição da estratégia de implementação
* **Validação** – descrição dos testes efetuados com apresentação com apresentação quantificada
* **Eficiência do protocolo de ligação de dados** –caraterização estatística da  eficiência do protocolo
* **Conclusões** –síntese da informação apresentada anteriormente reflexão sobre os objetivos de aprendizagem alcançados

# Arquitetura e Estrutura do código

O nosso trabalho foi construído com base no princípio da independência entre camadas, ou seja, o tornar o programa o mais modal possível. Para uma melhor implementação optamos por dividir em 3 parte:

## Data Link – Ligação de dados

Os ficheiros dataLink.h e dataLink.c representam o nível mais baixo da aplicação. É através deste que é estabelecida a ligação entre a porta de série e aplicação. Assim sendo, podemos concluir que é este que configura, inicia e termina a ligação e também o que escreve e lê os dados da porta de série, fazendo o tratamento dos erros, stuffing e destuffing.

int open\_serial\_port(char\* port, int whoCalls);

int open\_receiver(char\* port);

int open\_sender(char\* port);

int stuffing(unsigned char\* frame, int length);

int llopen(char\* port, int whoCalls);

int llread(char\* port, int whoCalls);

int llwrite(int fd, char\* buffer, int length);

int llclose(int fd, int whoCalls);

## App Link – Aplicação

Os ficheiros *AppLink.h* e *AppLink.c* representam o nível intermédio da aplicação, ou seja é o nível que faz a ponte de ligação entre os níveis mais baixo (*dataLink.h* e *dataLink.c*) e o mais alto (*interface*). Esta é também responsável pela comunicação da camada lógica com a interface do utilizador, é nesta camada que é feita a comunicação e transferência de dados, de acordo com os parâmetros que o utilizador define.

typedef struct{

int fileDescriptor;

int status;

}app;

int connection(char\* terminal, int whoCalls);

int receiveData();

int sendControlPackage(int state, FileInfo file, unsigned char \*controlPacket);

int sendDataPackage(unsigned char \*dataPacket, FILE \*fp, int sequenceNumber, int \*length);

int sendData();

# Casos de uso principais

A aplicação desenvolvida necessita apenas de um parâmetro, 0 ou 1 que representa a porta série a utilizar (/dev/ttyS0 ou (/dev/ttyS1). Depois de executar, a aplicação estabelece a comunicação e, no modo emissor, pede ao utilizador o ficheiro a transferir, o qual envia para o recetor. Durante o estabelecimento da comunicação, se estiver em modo recetor, espera que algum emissor inicie uma ligação.

# Protocolo de ligação lógica

O protocolo de ligação implementado tem como principais aspetos, pela seguinte ordem:

1. Configuração da porta de série;
2. Estabelecimento de ligação pela porta de série;
3. Transferência de dados pela porta de série (stuffing/destuffing);
4. Tratamento de erros durante a ligação.

Foi necessário implementar as seguintes funções:

* llopen();
* llclose();
* llread();
* llwrite().

## llopen() e llclose()

Estas são as funções responsáveis por iniciar e terminar a ligação pela porta de série. Para isso, a função llopen começa por testar quem a está a chamar, se o SENDER ou o RECEIVE e, a partir daí, chamar as funções adequadas para abrir a porta série e alterar as configurações da mesma para as pretendidas.

Na função llopen do SENDER, open\_sender(), é criada uma trama SET que é enviada para o recetor, entrando de seguida num ciclo de leitura para receber a resposta pretendida, a trama UA. Enquanto não a recebe, ativa um alarme de duração 3 segundos e, sempre que o alarme for desencadeado, conta como um timeout, tentando enviar outra vez a trama. Se exceder o número de timeouts máximo, a função termina com estado de erro, indicando que o llopen não conseguiu estabelecer comunicação e para tentar outra vez mais tarde. Se a aplicação for chamada como recetor, fica à espera até receber a trama SET o que quando sucede, envia a trama UA, estabelecendo corretamente a comunicação.

A função llclose, por sua vez, tenta terminar a ligação do lado do emissor, enviando uma trama DISC que espera pela resposta do recetor, que é outra trama DISC. Ao receber, envia uma trama UA, de modo a informar o recetor que recebeu a sua intenção de por término à comunicação. No lado do recetor, este espera pelo primeiro DISC referido acima e envia a resposta (trama DISC) e depois espera pela trama UA para e repõe as configurações anteriores da porta série, finalizando a ligação.

## llread() e llwrite()

A função llwrite é responsável por enviar os dados ao recetor, recebendo um parâmetro que pode ser um pacote de dados ou de controlo para o qual constrói a trama de informação correspondente que é posteriormente enviada ao recetor. Depois de enviar, o processo emissor espera pela resposta do recetor e, no caso de não a receber, volta a enviar dentro do número de tentativas estipulado (neste caso 3). Neste caso, a função termina com erro. A resposta recebida pode ser do tipo RR que indica o sucesso da transferência e determina a continuação da mesma, enviando a trama com o próximo número da sequência desde que não repetido, ou do tipo REJ que especifica a existência de erro de leitura e determina o reenvio da mesma trama.

A função llread, lê a trama de informação enviada pelo emissor e, após realizar o destuffing do pacote contido pela trama, é processada a informação, através de uma nova máquina de estados, de modo a que sempre que um byte errado é lido, volta ao estado original. Caso a trama recebida seja inicial o ficheiro é aberto para escrita de informação. Em resposta envia uma trama do tipo RR em caso de sucesso e REJ em caso de erro.

# Protocolo de aplicação

O protocolo de aplicação implementado tem como principais aspetos, pela seguinte ordem:

1. Geração e transferência dos pacotes de controlo de dados;
2. Leitura e escrita do ficheiro transferir.

## send\_Data() e receive\_Data()

# Validação

Para verificar a robustez da ligação, foram aplicados os seguintes testes:

1. Enviar um ficheiro.
2. Enviar um ficheiro e introduzir erros na ligação com um cabo de cobre.

A aplicação foi capaz de superar estes testes, verificando-se isto tanto pelos bytes do ficheiro estarem corretos, assim como pela demostração no ecrã do estado de envio.

No entanto, quando se realizaram os testes de envio de um ficheiro com interrupção da ligação da porta série, o processo de timeout não ocorria de maneira correta e o ficheiro não era completamente enviado.

# Eficiência do protocolo de ligação de dados

caraterização estatística da  eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido. A caracterização teórica de um protocolo Stop&Wait, que deverá ser usada como termo de comparação, encontra-se descrita nos slides de Ligação Lógica das aulas teóricas

# Conclusões

Por fim, concluímos que nem todos os objetivos propostos foram concluídos, uma vez que nem sempre o envio do ficheiro foi possível quando existiam erros de ligação da porta série. A nível geral, tivemos muita dificuldade em perceber como aplicar os conceitos explicados nas aulas e no guião, sentindo que este nem sempre era muito útil a responder às dúvidas que advinham das experiências realizadas a nível de código. Assim, e embora no final possamos dizer que compreendemos os conceitos necessários à realização do trabalho, bem como o sistema de independência de camadas, aonde demonstrámos a que a camada de aplicação se serve da camada de dados mas é independente do seu modo de agir, percebemos que a nossa aplicação necessita de muitas melhorias, especialmente na correção dos erros gerados pela abertura e fecho da ligação durante o processo de transferência.

# Anexo – Código fonte