1º **Trabalho Laboratorial - RCOM**

Diogo Guimarães do Rosário - 201806582

Henrique Melo Ribeiro - 201806529

**Sumário**

Este trabalho foi desenvolvido no âmbito da unidade curricular Redes de computadores. Este projeto consistia no desenvolvimento de uma aplicação

capaz de transferir dados de um computador para outro através de uma porta série assíncrona. A aplicação é resistente a erros na transmissão dos pacotes de dados

e desconexão da porta série.

A aplicação foi desenvolvida com sucesso, sendo possível transferir ficheiros entre dois computadores sem qualquer perda de informação.

**1.Introdução**

Este relatório tem o propósito de expor o modo como a nossa aplicação está organizada bem como o funcionamento desta.

O objetivo deste trabalho é implementar um protocolo de ligação de dados especificado no guião do trabalho, de modo a permitir transferência fiável de dados entre dois dispositivos

conectados pela porta série.

Assim o relatório estará organizado da seguinte forma:

2. Arquitetura e Estrutura do código - Demonstração dos blocos funcionais e interfaces e exposição das principais estruturas de dados, funções e sua relação com a arquitetura

3. Casos de uso principais - Identificação das sequências de chamada de funções

4. Protocolo de ligação lógica - Identificação dos principais aspetos funcionais, descrição da estratégia de implementação destes aspetos com apresentação de extratos de código

5. Protocolo de aplicação - Identificação dos principais aspetos funcionais, descrição da estratégia de implementação destes aspetos com apresentação de extratos de código

6. Validação - Descrição dos testes efetuados com apresentação dos resultados

7. Eficiência do protocolo de ligação de dados - Caraterização estatística da eficiência do protocolo

8. Conclusões - Síntese da informação apresentada nas secções anteriores

**2.Arquitetura e Estrutura de código**

O nosso projeto foi desenvolvido com duas camadas principais (protocolo e aplicação).

**2.1.Camada do protocolo**

A camado do protocolo está definida no ficheiro common.h e é a camada de nível mais baixo do nosso programa. É responsável pela comunicação entre os dois computadores através da porta série.

Para além disso, faz uso dos ficheiros writenoncanonical.c, writenoncanonical.h, noncanonical.c e noncanonical.h.

Esta separação foi feita devido a certas funções não serem necessárias dos dois lados do protocolo (escrita e leitura), como por exemplo a função de leitura de dados e a sua máquina de estados.

Foi utilizada a seguinte estrutura de dados para facilitar este processo:

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\**/

  int currentTry;                /*\*Número da tentativa atual em caso de falha\**/

  char frame[MAX\_SIZE\*2+7];          /*\*Trama\**/

  int frameSize;

};

**2.2. Camada de aplicação**

A camada da aplicação está definida nos ficheiros application.c e application.h e é a camada que está imediatamente acima do protocolo.

Esta camada é responsável pela leitura do ficheiro no lado da escrita e criar os pacotes de dados que serão transmitidos à camada do protocolo.

Foi utilizada a seguinte estrutura de dados para facilitar este processo:

struct applicationLayer

{

    int fileDescriptor; /\*Descritor correspondente à porta série\*/

    int status;         /\*TRANSMITTER | RECEIVER\*/

};

**3.Casos de uso principais**

A aplicação necessita de diferentes parâmetros dependendo se é recetor ou transmissor.

Do lado do transmissor é necessário o número da porta série a ser usada, a string "transmitter" e o nome do ficheiro.

Exemplo : ./application 0 transmitter pinguim.gif

Do lado do recetor é necessário o número da porta série a ser usada e a string "receiver".

Exemplo : ./application 0 receiver

O recetor quando iniciado fica à espera de um transmissor para iniciar a conexão. Após estabelecer esta conexão o ficheiro começa a ser transmitido pelo transmissor em pacotes de dados.

**4.Protocolo de ligação lógica**

O protocolo de ligação lógica tem como objetivos:

\* Configurar a porta série

\* Estabelecer a conexão entre as duas portas série

\* Transferência dos pacotes de dados após operações de *stuffing* e *destuffing*

\* Deteção de erros nas transmissões

**4.1. Configuração da porta série**

openWriter() e openReader()

Estas funções recebem como argumento a porta séria com a qual será estabelecida uma ligação e são invocadas na função na função **llopen()** sendo que esta função consegue distinguir qual o estado do programa (transmitter ou receiver) que a chama.

Pala além do referido anteriormente também são responsáveis por preencher a struct da camada de protocolo com os valores corretos.

Caso seja impossível alcançar a porta série desejada o programa lança uma mensagem e exit code -1.

**4.2 Estabelecer a conexão entre as duas portas série**

setupWriterConnection() e setupReaderConnection()

Estas funções são também invocadas pelo **llopen()** que lhes fornece o descritor das portas abertas pelas funções openWriter() ou openReader().

O estabelecimento da conexão começa com o programa emissor a enviar uma mensagem de controlo *set* que quando recebida e processada pelo programa recetor responde com uma mensagem *UA.*

Nestas funções também se dá *setup* ao sinal de alarm (SIG\_ALARM). O alarm serve como controlo nas comunicações, fazendo o emissor reenviar a última mensagem escrita caso este não receba resposta por parte do recetor. Existe uma ocasião em que os papeis estão invertidos, sendo esta quando os programas se estão a desconectar da porta série.

**4.3 Transferência dos pacotes de dados após operações de *stuffing* e *destuffing***

Esta transferência é garantida pelas função **sendInfo()** e **readInfo()**.

A função **sendInfo()** da camada de protocolo recebe o descritor da porta série, a mensagem e o seu tamanho, e retorna o número de bytes escritos. Nesta função são feitas operações de *stuffing* da mensagem de forma a garantir a comunicação correta entre programas

Depois de escrever para a porta série a função **readRR()** é invocada, fazendo o emissor esperar por uma resposta enviada pelo recetor. Se esta mensagem recebida for uma mensagem *receiver ready* o programa passa para o tratamento do próximo conjunto de dados que tem de enviar, caso seja de *reject* o programa reenvia a última mensagem e volta a ficar à espera de uma mensagem.

A função **readInfo()** responsabiliza-se por fazer a leitura da informação da porta série e de enviar a resposta correta ao programa emissor. Na função **readInfo()** são feitas operações de *destuffing* de forma a restaurar a mensagem recebida para o estado anterior ao *stuffing*  que esta sofreu.

**4.4 Deteção de erros nas transmissões**

A deteção de erros é feita ao mesmo tempo que a leitura da mensagem é feita, comparando a mensagem recebida com o **BCC** final (o **BCC** é igual à operação lógica ou exclusivo entre todos os bytes da informação), e verificando o **numero de série** da mensagem (varia entre 0 e 1 quando a mensagem recebida conter a informação correta).

No caso em que **BCC** está errado é enviada uma resposta *reject.* Caso o **número de** **série** esteja errado a informação recebida é considerada repetida e é enviada uma resposta *receiver ready* para o emissor enviar o próximo pacote de informação.

**5.Protocolo de aplicação**

O protocolo de aplicação tem como objetivos:

\* Gerar pacotes de controlo e de informação a partir da leitura do ficheiro

\* Leitura e escrita do ficheiro

\* Controlo do programa através das funções llread(), llwrite() ,llopen() e llclose();

**5.1. Criação pacotes de controlo e de informação a partir da leitura do ficheiro**

A criação de pacotes de controlo é realizada na função **buildControlPacket(),** função essa que recebe como argumentos o nome do ficheiro e o seu tamanho, bem como o byte do controlo (CONTROL\_START E CONTROL\_END no ficheiro application.h) e um buffer a preencher que corresponderá ao pacote completo. Este pacote será codificado no formato TLV (Type, length, value).

A criação de pacotes de informação é realizada na funcão **buildDataPacket(),** que receberá como parâmetros um *chunk* de tamanho MAX\_SIZE – 4 de modo a reservar 4 bytes para o byte de controlo, número de série e tamanho de bytes de informação. Este tamanho MAX\_SIZE corresponde ao tamanho que será lido de cada vez no ficheiro e está definido no ficheiro common.h

**5.2. Leitura e escrita do ficheiro**

A leitura do ficheiro só é realizada nos casos em que a aplicação é inicializada com o valor transmitter e é realizada num loop até o ficheiro ser inteiramente lido. Posteriormente a cada iteração do loop, é construído um pacote com esta informação e enviada para o protocolo de dados através da função **llwrite().** Do lado do recetor, a informação é lida e guardada num array com o tamanho do ficheiro alocado aquando da leitura do primeiro pacote de controlo, visto que neste pacote é recebida a informação do nome e tamanho do ficheiro.

No final do programa este array é inteiramente escrito para o ficheiro usando **fwrite(),** criando assim o ficheiro com toda a informação recebida.

**5.3. Controlo do programa através das funções llread(), llwrite() ,llopen() e llclose();**

Incialmente, ambos os programas invocam **llopen()**, de modo a abrir a porta série e iniciar a comunicação destas através das sequências *set* e *ua*. Posteriormente é feito um loop na qual o transmissor lê o ficheiro e invoca **llwrite()**, enquanto que o recetor apenas invoca **llread()** também num loop controlado pela flag finished. Aquando da leitura do pacote de controlo final, verifica-se que este pacote é igual ao pacote controlo inicial e caso seja, a flag finished será posta a true, acabando o loop de leitura.

Posteriormente, é invocado **llclose()** nos 2 lados da aplicação, desconectando a porta série e escrevendo o ficheiro recebido do lado do recetor.

**6. Validação.**

Para verificar a integridade do nosso código, o programa foi sujeito a diversos testes:

-Passagem de um ficheiro extensão .gif (11 KB)

-Passagem de um ficheiro com extensão .mp4 (13 MB)

-Interrupção do programa através de time-out.

-Desligar a o cabo da porta série aquando da transferência de um ficheiro sem a ligar novamente.

-Desligar a o cabo da porta série aquando da transferência de um ficheiro, voltando a ligá-lo novamente.

-Criação de erros na ligação da porta série.

-Variação do tamanho máximo da trama de informação (testes realizados com 250, 500 e 1000 bytes).

-Variação do baud rate da porta série.

Perante todos estes testes o programa conseguiu terminar com sucesso sendo posteriormente verificada esta integridade usando o comando *diff* nativo do Linux.

**7. Eficiência do protocolo de ligação de dados**

**7.1 Variação de erros**

Como se pode observar no seguinte gráfico, os erros causados no BCC1 e BCC2 (erros causados usando valores random) têm um impacto grande na eficiência do programa.

Após uma análise mais detalhada é possível observar que os erros que causam maior impacto na eficiência são os erros causados no BCC1 (cada erro causa por volta de 1 segundo de atraso visto que o programa terá de esperar pelo sinal de alarme para reenviar a informação) enquanto os erros causados pelo BCC2 quase que não têm impacto na eficiência do programa, visto que quando isto acontece, o recetor envia uma mensagem de *reject* que faz com que o emissor reenvie a mensagem imediatamente.

Gráfico obtido usando um ficheiro de tamanho 10968 bytes, baudrate 38400 e chunk size de 250. O eixo x corresponde à taxa de erros causados no BCC1 (erros no BCC do cabeçalho) e BCC2 (erros no BCC de informação) e o y ao FER.

**7.2 Variação do tamanho das tramas**

Como se pode verificar pelo gráfico a baixo, ao aumentar o tamanho da trama, a eficiência do programa aumenta.

**8. Conclusão**

O fundamento deste trabalho é a passagem de informação entre processos garantindo a sua integridade.

Algo muito importante para este fim é a separação de tarefas entre a camada de aplicação que trata da leitura e escrita de ficheiros e da camada de protocolo que trata da comunicação entre processos através de uma porta série, sem que nenhuma precise de saber os detalhes da outra, conseguindo realizar a sua funcionalidade com os dados fornecidos.

Em suma, pensamos que o objetivo global deste trabalho foi alcançado. Apesar de no início do projeto sentirmos que os fundamentos necessários para realizar o projeto fossem muito complexos após algumas leituras do enunciado e trocas de ideias com colegas achamos que conseguimos entender bem o necessário e colocar este conhecimento em prática.