

**1º Trabalho Laboratorial**

Ligação de Dados

Grupo 7

**Trabalho realizado por:**

Hugo Gomes – up202004343

João Moreira – up202005035

**Unidade Curricular:**

RC

**Ano Letivo:**

2022/2023

**Sumário**

No âmbito da unidade curricular de Redes de Computadores, foi nos proposto a implementação de um protocolo de ligação de dados, para a transferência de ficheiros de um computador para outro, através de uma porta série.

Assim, podemos afirmar que concluímos todos os objetivos propostos, obtendo uma aplicação robusta capaz de transferir ficheiros sem perda de dados.

**Introdução**

Este trabalho tem como principal objetivo a implementação de um protocolo de ligação de dados de acordo com as informações descritas no guião.

Neste guião, foi nos pedido que a nossa implementação seguisse algumas características especificas, entre as quais: independência entre camadas, isto é, separação total entre a camada de ligação de dados e a camada de aplicação, tendo, esta última, total desconhecimento dos detalhes da camada de ligação, usando-a apenas como um serviço, confirmação e controlo de erros através de uma variante de Stop and Wait, utilização da técnica de byte stuffing entre outros.

Quanto ao relatório, este tem como objetivo a exposição da componente teórica deste projeto, tendo a seguinte estrutura:

* Arquitetura- blocos funcionais e interfaces.
* Estrutura do código – APIs, principais estruturas de dados, principais funções e a sua relação com a arquitetura.
* Casos de uso principais – identificação e sequências de chamada de funções.
* Protocolo de ligação lógica - identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes aspetos.
* Protocolo de aplicação - identificação dos principais aspetos funcionais, descrição da estratégia de implementação destes aspetos.
* Validação - descrição dos testes efetuados.
* Eficiência do protocolo de ligação de dados - caraterização estatística da eficiência do protocolo, efetuada recorrendo a medidas sobre o código desenvolvido.
* Conclusões - Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

**Arquitetura**

O projeto está organizado em dois blocos funcionais: um associado ao computador que envia o ficheiro (sender) e outro ao computador que recebe o ficheiro (receiver). Os ficheiros receiver.c e receiver.h contêm o código do receiver enquanto os ficheiros sender.c e sender.h o código correspondente ao sender. Em cada um destes blocos funcionais, existe uma separação clara entre a camada de aplicação e a camada de ligação de dados.

A camada de ligação de dados tem como objetivo fornecer um serviço de comunicação fiável, entre dois sistemas por meio de um canal. Nesta camada, encontramos funções genéricas do protocolo de ligação de dados que fornecem as funcionalidades de sincronismo de tramas (framing), numeração de tramas, controlo de erros, controlo de fluxo e estabelecimento/término da ligação.

A camada de aplicação utiliza as funções da camada de ligação de dados como um serviço para transferência da informação. Esta é responsavel por enviar o ficheiro, dividindo-o em pacotes de informação que são passados à camada de ligação de dados.

**Estrutura do código**

O código encontra-se dividido em dois ficheiros: o sender.c que corresponde ao writer e o receiver.c que corresponde ao reader. Ambos estes ficheiros chamam funções da camada da aplicação que, por sua vez, chama funções da camada da ligação de dados.

**Receiver:**

Funções da camada de ligação:

* **LLOPEN()** - Recebe trama SET e envia trama UA;
* **LLREAD()** - Lê trama de informação e realiza destuffing;
* **LLCLOSE()** - Recebe trama DISC e envia trama DISC;

Funções camada de aplicação

* **createFile() -** Cria o ficheiro através da informação recebida;
* **isEndPacket() -** Verifica se o packet recebido é o END packet;
* **removeControlHeader() -** Remove os bytes associados ao controlo do packet obtendo apenas os dados;
* **main() -** Função principal da camada de aplicação que utiliza todas as outras funções para abrir a ligação, ler um packet de cada vez até receber o END packet, construir o ficheiro com os packets recebidos e fechar a ligação;

Variáveis globais

* **volatile int STOP;**
* **int expectedFrame;**
* **struct termios oldtio;**
* **struct termios newtio;**

Macros pertinentes:

* **BAUDRATE**

**Sender:**

Funções da camada de ligação de dados:

* **LLOPEN()** - Envia trama SET e recebe trama UA;
* **LLWRITE()** - Constrói uma trama de informação com os dados recebidos como argumento, realiza byte stuffing e tenta enviar esta mesma trama;
* **LLCLOSE()** - Envia trama DISC, recebe trama DISC e envia trama UA;

Funções da camada de aplicação:

* **openFile()** - Abre o ficheiro, calcula o tamanho do ficheiro e lê todos os bytes do mesmo para um buffer;
* **createControlPacket()** - Cria START packet, caso start = True. Caso contrário, criar END packet;
* **createPacket()** - Cria packet de informação, através do buffer de bytes do ficheiro;
* **addHeaderPacket()** - Adiciona header com campo de controlo, número de sequência, número de octetos a enviar ao packet de dados;
* **main()** - Função principal da camada de aplicação que utiliza todas as outras funções para abrir a ligação, abrir o ficheiro, construir os packets de controlo/informação, enviá-los através da camada de ligação de dados e fechar a ligação;

Variáveis globais

* **int alarmEnabled;**
* **int alarmCount;**
* **int UAreceived;**
* **int currentFrame;**
* **struct termios oldtio;**
* **struct termios newtio;**

Macros pertinentes

* **BAUDRATE**
* **TIMEOUT**
* **MAX\_SENDS**

**Casos de Uso Principais**

A nossa aplicação permite ao utilizador transmitir um ficheiro de um computador para outro, através de uma porta série. Em primeiro lugar, o utilizador precisa de compilar a aplicação a partir de um Makefile. De seguida, o computador que pretende enviar o ficheiro (sender) pode executar o programa da seguinte forma: ./sender [porta-série] [nome ficheiro] Ex: ./sender /dev/ttyS10 pinguin.gif. Por outro lado, o computador que pretende receber o ficheiro (receiver) pode executar o programa da seguinte forma: ./receiver [porta-sérire] Ex:/dev/ttyS10.

O computador receiver deverá ser o primeiro a ser executado para esperar que o transmiter envie as tramas de estabelecimento de ligação. Caso contrário, o transmiter iniciará as tentativas de envio das tramas e terminará quando o número de tentativas for excedido.

Após a ligação ser estabelecida, por cada packet enviado, uma mensagem de envio ou receção surge na consola. Caso ocorram erros, as seguintes mensagens surgem na consola:

* + - Erro de BCC1 -> Error in the protocol;
    - Erro de BCC2 -> Error in the data;
    - Frame duplicado -> Duplicate Frame;
    - Reject recebido -> REJ received;

****Protocolo de ligação lógica****

Máquinas de Estado:

Antes de abordamos a nossa implementação das funções principais é necessário explicar como implementamos as máquinas de estado. ///TALVEZ METER FOTO DAS MAQUINAS DE ESTADOS:

No recetor, criamos uma máquina de estados geral que é capaz de receber uma trama UA ou uma trama DISC. A função receiveControlWord recebe o file descriptor da porta série e um unsigned char que representa quais das tramas de supervisão estamos a tentar ler UA ou DISC.

No emissor, criamos uma máquina de estados geral que é capaz de receber uma trama UA, REJ, RR ou DISC. A função receiveControlWord recebe o file descriptor da porta série e um apontador para um unsigned char onde será devolvido o tipo de trama recebido.

**LLOPEN**

int LLOPEN (int fd);

Esta função é responsável pela fase de estabelecimento da ligação. No recetor, utilizando a função receiveControlWord tentamos ler a trama SET. De seguida, caso a receção seja confirmada, enviamos a trama UA.

No emissor, a trama SET é enviada e um temporizador é ativado. Caso o temporizador dispare, um reenvio da trama SET é executado. As tentativas de reenvio terminam após um número pré-definido de tentativas ser excedido ou a trama UA ser recebida. Caso este ciclo termine pelo número máximo de tentativas, a função retorna FALSE, caso contrário retorna TRUE.

**LLREAD**

int LLREAD(int fd, unsigned char \* messageReceived);

Esta função, apenas, está presente no recetor e é responsável por fazer a leitura de uma trama e o destuffing da mesma. De modo a fazer a leitura caracter a caracter, criamos uma maquina de estados. //FOTO MAQUINA DE ESTADOS??

Esta máquina de estados inicia no estado 0. Neste estado, verificamos a receção da FLAG. Caso se verifique, passamos para o estado um onde confirmamos a receção do A e, consequentemente, saltamos para o estado dois. No estado dois, verificamos o campo de controlo C e guardamos este para, mais tarde, no estado três, fazermos a verificação do BCC1. Caso o BCC1 esteja correto, passamos ao estado 4 onde vamos começar a receber os dados. Se o caracter recebido for uma flag, significa que a trama chegou ao fim e saltamos para o estado 6, caso o caracter seja um ESC teremos que realizar o destuffing, executado no estado 5. Caso contrário, adicionamos o caracter recebido ao buffer onde guardamos a mensagem recebida. Por fim, no estado 6, é calculado o BCC2 para os dados que recebemos e comparado com o caracter recebido antes da FLAG. Caso este se confirme, enviamos um received ready, caso contrário, enviamos um reject.

Esta função retorna o tamanho da mensagem recebido, caso todas as condições estejam corretas. Caso alguma falhe, retorna -1. O apontador para o buffer de dados é retornado no parâmetro messageReceived.

**LLWRITE**

int LLWRITE (int fd, unsigned char\* msg, int size);

Esta função apenas está presente no emissor e é responsável por construir uma trama com a mensagem recebida, fazer o stuffing da mesma e enviá-la.

Primeiro, é criado um buffer com o tamanho da mensagem mais 6 bytes. Estes 6 bytes são utilizados para os campos de controlo do protocolo de ligação de dados (FLAG,A,C,BCC1). De seguida, realizamos stuffing nos dados, calculamos o BCC2 dos dados sem stuffing e fazemos stuffing do BCC2. Por fim, adicionamos a FLAG final e finalizamos a criação da trama.

Deste modo, o passo seguinte é enviar a trama. O envio da trama de informação é semelhante ao envio da trama de supervisão, isto é, tentamos enviar a trama, iniciamos um temporizador e iniciamos a state machine de leitura. Caso a palavra de controlo recebida seja um Received Ready, a função termina e retorna o tamanho da mensagem enviada. Caso seja um Reject, se este foi provocado pelo envio de um frame duplicado, a função termina alterando o número da trama a enviar. Caso contrário, o temporizador é reiniciado, o número de tentativas de envio volta a zero e uma nova tentativa de envio é efetuada. As tentativas de envio terminam após um número de tentativas máximo ser excedido ou um Received Ready ser recebido.

**LLCLOSE**

int LLCLOSE (int fd);

Esta função é responsável pela fase de terminação da ligação. No recetor, utilizando a função receiveControlWord, tentamos ler a trama DISC. De seguida, caso a receção seja confirmada, enviamos a trama DISC.

No emissor, a trama DISC é enviada e um temporizador é ativado. Caso o temporizador dispare, um reenvio da trama DISC é executado. As tentativas de reenvio terminam após um número de tentativas ser excedido ou a trama DISC ser recebida. Após a receção do disc ou o número de tentativas de reenvio ser atingido, um UA é enviado.

Em ambos os programas, é reposta a configuração inicial da porta série, restaurando os valores iniciais alterados no LLOPEN.

****Protocolo de aplicação****

Em ambas as máquinas, no início da função main, é feita uma verificação dos argumentos recebidos pelo utilizador. Caso haja algum erro, uma mensagem com um exemplo de utilização é apresentada.

No emissor, o programa começa por abrir o ficheiro para envio, calcula o tamanho do mesmo e lê todos os seus bytes. A execução continua com a chamada da função LLOPEN para estabelecer a ligação. De seguida, é necessário criar o START PACKET. Esta ação é realizada pela função createControlPacket com o parâmetro start a TRUE. O retorno desta função é passado à função LLWRITE. Enquanto o ficheiro não terminar, a função createPacket é chamada para criar um packet com uma parte do mesmo. Através da função addHeaderPacket, o header da camada de aplicação é concatenado ao packet de dados anterior e este é passado à função LLWRITE para envio. Por fim, novamente utilizando a função createControlPacket, mas com o paramentro start a FALSE, o END PACKET é criado e enviado com o LLWRITE.

No recetor, primeiramente, a função LLOPEN é chamada para o estabelecimento da ligação. De seguida, utilizamos a função LLREAD para receber o START PACKET e retirar o tamanho do ficheiro do mesmo. Posteriormente, enquanto o END PACKET não for recebido, lemos um packet, retiramos o header deste através da função removeControlHeader e adicionamo-lo ao buffer com a informação recebida. Por fim, chamamos a função createFile que cria um ficheiro com toda a informação recebida.

No fim dos dois programas, a função LLCLOSE é evocada para iniciar a fase de término da ligação.

****Validação****

De modo a validar a robustez do nosso programa, efetuamos um conjunto de testes. Começamos por interromper a ligação da porta série, fazendo com que ocorra um timeout no emissor e uma tentativa de reenvio. De seguida, testamos a forma como o programa lida com cada tipo de erro, quer gerando curto circuito na porta série, quer introduzindo erros simulados, através das funções errorBCC1 e errorBCC2. Por fim, alteramos o tamanho dos pacotes e o tamanho do ficheiro. Todos estes testes foram realizados com sucesso.

**Eficiência do protocolo de ligação de dados**

O cálculo da eficiência é obtido dividindo o débito(R) pela capacidade de transmissão. Para calcular o débito R, medimos o tempo de execução do nosso programa e dividimos pelo tamanho do ficheiro a enviar. Deste modo, elaboramos 3 gráficos variando vários fatores que influenciam a eficiência.

No primeiro, variamos a percentagem de erros gerados e concluímos que, com o aumento da percentagem de erros, a eficiência diminui.

No segundo, variamos o tamanho das tramas de informação e concluímos que, com o aumento do tamanho das tramas, a eficiência aumenta.

No terceiro, variamos a capacidade de ligação (baudrate) e concluímos que a eficiência diminui ligeiramente com o aumento do baudrate.

**Conclusões**

**O tema deste trabalho é o protocolo de ligação de dados, que consiste em fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão, neste caso em concreto, um cabo série.**

**Adicionalmente, foi dado a conhecer o termo independência entre camadas, e cada um dos blocos funcionais da arquitetura da aplicação desenvolvida, writer e reader, cumpre esta independência. Na camada de ligação de dados, não é feita qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de Informação. No que diz respeito à camada de aplicação, esta não conhece os detalhes do protocolo de ligação de dados, apenas a forma como o serviço é acedido.**

**Em suma, o trabalho foi concluído com sucesso, tendo-se cumprido todos os objetivos. A sua elaboração contribuiu positivamente para um aprofundamento do conhecimento, tanto teórico como prático, do tema em questão.**

**Anexos**

Sender.h

Text

Description automatically generated

Text

Description automatically generated

Sender.c

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Receiver.h

Text

Description automatically generated

Text

Description automatically generated

Receiver.c

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated Text

Description automatically generated

Text

Description automatically generated