CAPA

* **Título, Autores**
* **Sumário**

*dois parágrafos: um sobre o contexto do trabalho;*

*outro sobre as principais conclusões do relatório*

Introdução

1. *Indicação dos objetivos do trabalho e do relatório;*

O trabalho apresentado foi desenvolvido em linguagem de progamação C e tem como objetivo a implementação de um protocolo de ligação de dados em Linux, através da utilização de portas série RS-232 (comunicação assíncrona). Este protocolo fornece um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão neste caso, um cabo série e é testado com uma aplicação simples de transferência de ficheiros entre o Emissor e o Recetor, sendo todo o seu processo de construção, desenvolvimento e implementação explicado ao longo deste relatório.

1. *descrição da lógica do relatório com indicações sobre o tipo de informação que poderá ser encontrada em cada uma secções seguintes.*

Arquitetura e Estrutura do código

Em termos de fluxo do código, é o ficheiro main.c que inicia todo o processo de transmissão de dados. É neste ficheiro que são definidas as especificações da porta série e do alarme.

#define BAUDRATE 9600

#define N\_TRIES 3

#define TIMEOUT 4

"Starting link-layer protocol application\n"

           "  - Serial port: %s\n"

           "  - Role: %s\n"

           "  - Baudrate: %d\n"

           "  - Number of tries: %d\n"

           "  - Timeout: %d\n"

           "  - Filename: %s\n",

A função main invoca a função applicationLayer com as especificações pré definidas e as de input, sendo aqui o ponto de partida para o resto do programa.

applicationLayer(serialPort, role, BAUDRATE, N\_TRIES, TIMEOUT, filename);

No nosso trabalho, é notório a separação do código referente ao emissor e ao recetor. Apesar de existirem funções utilizadas só pelo emissor e outras só pelo recetor, há também funções partilhadas pelos dois, em que o código utilizado por cada um é diferenciado por expressões condicionais.

A distinção entre camadas é percetível pela criação do ficheiro application\_layer.c, que contém as funções pertencentes à camada de Aplicação, juntamente com o ficheiro link\_layer.c, que agrupa as funções referentes ao Protocolo de Ligação de Dados.

Casos de uso principais

Como explicado anteriormente, o programa começa pelo main que chama o applicationLayer e a partir daí segue caminhos distindos dependendo se é o emissor ou recetor. A seguir promenoriza-se a sequência de chamada de funções para cada um dos casos possíveis.

Caso I – Emissor

No caso do emissor, a função base chamada será o applicationTx, onde abrimos o ficheiro de onde vamos ler os dados para enviar ao recetor e logo depois começamos por fazer o set up: preenchemos a struct LinkLayer com os parametros definidos para o alarme e chamamos llopen para configurar a porta série através do openPort e iniciar a conexão. Aqui é enviado o SET frame através da função writeCtrlFrame, que configura a flag e os campos de endereço, controlo e de proteção, e espera-se pela confirmação de recebimento, o UA frame. Após esta etapa, não ocorrendo nenhum erro, a conexão inicial está estabelecida.

O passo seguinte é iniciar a transferência de dados. Como estamos no caso do emissor, a função chamada será a applicationTx. Esta função abre o ficheiro para o qual queremos escrever e começa por configurar o pacote de controlo através do writeCtrl para sinalizar o início da transmissão. Apesar de neste tipo de pacote só ser obrigatório ter um campo com o tamanho do ficheiro, optamos também por inserir um campo com o nome do ficheiro. Ainda dentro desta função é chamado o llwrite para enviar a trama de informação que contém o pacote de controlo configurado no passo anterior. A configuração do cabeçalho desta trama é efetuada pela função prepareWrite que configura a flag e os campos de endereço, contolo e de proteção, assim como efetua o mecanismo de byte stuffing, invocando a função stuff do utils.c. Após todas estas etapas estarem concluídas, voltamos ao llwrite onde finalmente podemos escrever para o ficheiro e esperar pela receção da trama de confirmação. Ao receber uma trama sem erros e na sequência correta, estamos prontos para começar a enviar dados.

Como queremos iniciar a tranferência de dados, após o envio com sucesso do pacote de controlo inicial, voltamos a estar dentro do applictionTx e enquanto houver dados para enviar, procedemos à configuração do pacote de dados para posterior envio. Para isso, chamamos a função configureDataPackage que preenche a trama com o campo de controlo, o número de sequência, o campo de dados com o que foi lido do ficheiro, e o campo com o tamanho deste e, de seguida, invocamos o llwrite novamente, para enviar a trama de informação que contém o pacote de dados configurado no passo anterior. Após isto, é chamada a função printBar que vai imprimir para o terminal a quantidade de bytes que já foram enviados em suma, por cada iteração. Este conjunto de intruções serão repetidas até que os dados a enviar acabem, ou caso exista algum erro na trasmissão das tramas.

Para sinalizar o térmito da transmissão, utiliza-se novamente a função writeCtrl que vai mais uma vez configurar o pacote de controlo. Como no inicio da transmissão, o llwrite é invocado para enviar a trama de informação contendo este pacote no seu campo de dados. Após toda a configuração do cabeçalho da trama e da passagem pelo mecanismo de byte stuffing, a trama é finalmente enviada e, desta forma, está assegurado o fim da transmissão de dados, pelo que o ficheiro já poderá ser fechado.

Por fim, voltando à função inicial applicationLayer, só nos resta invocar o llclose que, como estamos no caso do emissor, vai enviar o DISC frame através da função writeCtrlFrame e esperar pela confirmação de recebimento. Caso haja essa confirmação, o writeCtrlFrame é novamente invocado para enviar o UA frame e concluir a terminação da transmissão. Finalizando, as definições iniciais da porta série são restaurados e esta é fechada.

Caso II – Recetor

Já no caso do recetor, a função invocada será o applicationTx, onde se abre o ficheiro para onde vamos escrever os dados enviados pelo emissor e, seguidamente, a função setUp é novamente invocada para preencher a struct LinkLayer e chamar llopen para configurar a porta série com a função openPort, iniciando a conexão. Como já explicado no emissor, é enviada a SET frame, pelo que, o recetor, nesta fase da conexão, vai esperar pelo seu envio e enviará uma trama de confirmação (UA frame) de volta para o emissor, caso não ocorra nenhum erro. Após este processo, o estabelecimento da conexão foi efetuado com sucesso.

Depois da conexão inicial, voltamos ao applicationRx para dar início à transferência de dados. Para isso, entramos num ciclo de instruções, que se irá repetir enquanto houver dados para ler da porta série. Pormenorizando, dependendo de algumas especificações, poderemos ler pacotes de controlo ou de dados. No caso de ser lido um pacote de controlo inicial, executam-se técnicas para armazenar o tamanho do ficheiro e o seu nome com a ajuda da função receiveCtrl, que serão usados para futura comparação como o pacote de controlo final e verificar se não ocorreu nenhum erro inesperado. Após a receção do pacote de controlo inicial, passamos à receção de pacotes de dados: depois de verificar se o pacote recebido é o suposto, ou seja, se não existiu perda de pacotes, os dados retirados do ficheiro com dados a enviar pelo emissor são agora escritos pelo recetor para o ficheiro respetivo. Sempre que é recebido um pacote de dados, é impresso para o terminal, através da função printBar, a quantidade de bytes que já foram escritos. No final, espera-se pela receção do pacote de controlo final, e, caso as informações sobre o ficheiro coincidam, este é fechado sem nenhum erro ter ocorrido.

Por fim, voltando à função inicial applicationLayer, só nos resta invocar o llclose que, como estamos no caso do recetor, apenas vai restaurar as definições iniciais da porta série assim como a irá fechar. Assim a terminação da trasmissão foi efetuada com sucesso.

Protocolo de ligação lógica

Aqui especificaremos alguns aspetos funcionais pertencentes ao protocolo de ligação de dados que foram implementados no nosso trabalho.

Neste protocolo, a transmissão é organizada em tramas, que podem ser de três tipos: Informação, Supervisão e Não Numeradas. Apenas as tramas de informação possuem um campo para transporte de dados (gerados pela aplicação), contudo todas elas têm um cabeçalho com um formato comum.

As tramas de Supervisão, assim como as Não Numeradas, são construidas pelo writeCtrlFrame.

int writeCtrlFrame(int fd, unsigned char ctrl, unsigned char addr)

{

    unsigned char buf[BUF\_SIZE + 1] = {0};

    buf[0] = FLAG;         // flag: 0x7E

    buf[1] = addr;         // campo de endereço: 0x03 ou 0x01

    buf[2] = ctrl;         // campo de controlo: SET, DISC, UA, RR, REJ

    buf[3] = addr ^ ctrl;  // campo de proteção de cabeçalho

    buf[4] = FLAG;         // flag: 0x7E

    buf[5] = '\0';

    return write(fd, buf, 5);

}

A delimitação de tramas é feita por meio de uma sequência especial de oito bits que designamos de flag (01111110 ou 0x7E). Uma trama pode ser iniciada com uma ou mais flags o que deve ser tido em conta pelo mecanismo de receção de tramas.

O mecanismo de byte stuffing está recriado no nosso código pelo uso das funções stuff e deStuff.

/\*\*

 \* Stuff a sequence of bytes.

 \* @param buf byte array to be stuffed.

 \* @param bufSize size of the buffer.

 \* @return size of the new buffer.

\*/

int stuff(const unsigned char \*src, unsigned char\* dest, int bufSize);

/\*\*

 \* De-Stuff a sequence of bytes.

 \* @param buf byte array to be de-stuffed.

 \* @param bufSize size of the buffer.

 \* @return size of the new buffer.

\*/

int deStuff(unsigned char \*dest, int bufSize);

Tal como seria esperado, stuff substitui a ocorrência da flag 0x7E pelo octeto de escape 0x7D seguido do byte 0x5E, assim como substitui o octeto de escape 0x7D pela sequência de dois bytes 0x7D 0x5D. O processo contrário é efetuado pelo deStuff que ao encontrar o octeto de escape 0x7D, verifica qual o byte que lhe sucede e, substitui os dois bytes pela flag 0x7E, caso o segundo byte seja 0x5E, ou pelo octeto de escape 0x7D, caso o segundo byte seja 0x5D. Assim, asseguramos a transparência garantindo que o protocolo garante a transmissão de dados independente de códigos.

A atribuição do endereço ao cabeçalho da trama vai variar da seguinte forma: em Comandos enviados pelo Emissor e Respostas enviadas pelo Recetor o campo de endereço é preenchido com o octeto 00000011 (ou 0x03); em Comandos enviados pelo Receptor e Respostas enviadas pelo Emissor é o octeto 00000001 (ou 0x01) que preenche esse campo.

É importante salientar que o campo de controlo recebe oito bit diferentes dependendo do tipo de trama.



Por isso, o parametro ctrl é chamado com 0x03, 0x0B e 0x07 para o SET, DISC e UA respetivamente, mas quando se refere a uma trama RR, por exemplo, o número da trama (R – pode ser 0 ou 1) em questão é deslocado sete bits para a esquerda, para ficar na posição correta, e é feito um OR bit a bit com 0x05, para o ctrl ficar com o formato R0000101. O processo idêntico é feito para o caso de ser uma trama REJ.

writeCtrlFrame(fd, RR | (!frameNumber << R\_CTRL\_SHIFT), ADDR\_T);

As tramas são também protegidas por um código detetor de erros. Nas tramas de Supervisão e Não Numeradas, já que não existe o transporte de dados, existe apenas proteção simples da trama. Contudo, nas tramas de Informação existe proteção dupla independente, no cabeçalho e no campo de dados, o que permite usar um cabeçalho válido, mesmo que ocorra erro no campo de dados. No nosso trabalho, as tramas de Informação são estruturadas no prepareWrite e, abaixo, apresentamos um exemplo de como a proteção do campo de dados é efetuada.

int prepareWrite(const unsigned char\* buf, unsigned char\* dest, int bufSize) {

    unsigned char copy[BUF\_SIZE + 1] = {0};

    unsigned char to\_stuff[BUF\_SIZE + 1] = {0};

    dest[0] = FLAG;

    dest[1] = ADDR\_T;

    dest[2] = frameNumber << I\_CTRL\_SHIFT;

    dest[3] = ADDR\_T ^ dest[2];

    unsigned char bcc = 0;

    // for loop does the parity over each of the bits of 'the data octets' and the 'BCC'

    for (int j = 0; j < bufSize; j++) {

        bcc ^= buf[j];

        to\_stuff[j] = buf[j];

    }

    to\_stuff[bufSize] = bcc;

// continua no código..

**Protocolo de aplicação**

identificação dos principais aspetos funcionais;

descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.

**Validação**

descrição dos testes efetuados com apresentação quantificada dos resultados, se possível.

**Eficiência do protocolo de ligação de dados**

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

1. **Conclusões**

síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados.