PROTOCOLO DE LIGAÇÃO DE DADOS

Turma:	3LEIC09
José Silva	up201906576
Pedro Carneiro	up201906293

Sumário

No âmbito da unidade curricular de Redes de Computadores, o presente relatório tem por objetivo descrever e complementar o primeiro trabalho laboratorial, a realização de um protocolo de ligação de dados.

A realização deste relatório permitiu organizar de forma clara todos as componentes envolvidas no projeto e possibilitou uma aprendizagem mais concisa dos conceitos abordados.

Introdução

O trabalho tinha como objetivo a realização de um protocolo de ligação de dados que estabelece uma comunicação assíncrona, entre dois computadores, para a transmissão de um ficheiro, recorrendo a uma Porta Série RS-232.

O seguinte relatório pretende realizar uma breve explicação da temática envolvida, identificando os principais objetivos do trabalho laboratorial. Deste modo, está dividido em secções principais, a fim de explicar, de forma concisa, os principais aspetos funcionais e estruturais do protocolo realizado.

Assim, este relatório está organizado da seguinte forma:

- **Arquitetura** - Identificação dos blocos funcionais e interfaces;
- **Estrutura do código -** Demonstração das APIs, principais estruturas dos dados, principais funções e sua relação com a arquitetura;
- **Casos de Uso Principais –** Identificação dos mesmos e demonstração das sequências de chamada de funções;
- **Protocolo de ligação lógica –** Identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código;
- **Validação e Eficiência de ligação de dados –** Descrição dos testes efetuados com apresentação quantificada dos resultados
- **Conclusão –** Síntese do projeto realizado e reflexão sobre os principais objetivos de aprendizagem alcançados.

Arquitetura

O código está divido em duas camadas que permitem a correta funcionalidade do protocolo: a camada de ligação de dados e a camada de aplicação. Cada camada têm um ficheiro *source* associado, *link_layer.c* no caso da camada de ligação de dados e application_layer.c na camada de aplicação. Para além disso, cada camada possuí um header, link_layer.h e application_layer.h, respectivamente, que incluem a declaraçõ da API do protocolo e os parâmetros relativos à porta série. Foi ainda criado, um header file *link_layer_utils.h* que inicializa as funções, variáveis e estruturas utilizadas no código *link_layer.c*.

Estrutura do Código

A camada de ligação de dados é a camada responsável pelo estabelecimento de ligação e, portanto, tem todas as funções que sustentam a consistência do protocolo, como o tratamento de erros e o envio de mensagens de comunicação. Adicionalmente, é nesta camada que se estabelece a conexão e comunicação com a porta série.

Deste modo, esta camada é representada através de uma estrutura de dados onde é guardado a porta série utilizada, o *baudrate*, o tempo esperado até ao reenvio de uma trama, o número máximo de tentativas de reenvio e ainda a varável *role* que permite a distinção entre o transmissor e o recetor.

```
typedef struct
{
    char serialPort[50];
    LinkLayerRole role;
    int baudRate;
    int nRetransmissions;
    int timeout;
} LinkLayer;
```

```
int llopen(LinkLayer connectionParameters);

int llwrite(const unsigned char *buf, int bufSize);

int llread(unsigned char *packet);

int llclose(int showStatistics);
```

Casos de Uso Principais

Os principais casos de uso desta aplicação é a transferência de um ficheiro previamente selecionado, através da porta série, entre dois computadores, o emissor e o recetor.

O recetor deverá ser iniciado primeiro para esperar que o emissor inicie a ligação. Caso contrário, o emissor nunca irá conseguir estabelecer a ligação e, quando o número máximo de tentativas for excedido, o programa termina.

Sequência de funções no emissor

1. llopen - Estabelece a ligação através da porta de série:

a. *Set_Serial_Port* – Configura a porta de série;

b. *Open_Transmitter* – Invoca a função ***Send_And_Receive_Frame*** e verifica se o envio e receção das tramas foi realizado corretamente;

c. Send_And_Receive_Frame - Envio da trama SET (início da conexão) e aguarda receção da trama UA (Unnumbered Acknowledgment);

d. Check_State_Machine – Verifica se a trama recebida é a pretendida (UA).

2. llwrite - Responsável pelo encapsulamento e *Stuffing* dos dados. A trama, do tipo I, depois de criada é enviada para o recetor. Na ausência de uma resposta de confirmação por parte do recetor ou na receção de uma trama do tipo REJ, a trama é reenviada.

a. Send_And_Receive_Frame – Envio da trama encapsulada (cabeçalho e campo de dados) e receção da trama RR com o número de sequência correto.

3. llclose – Encerramento da ligação.

a. *Close_Transmitter* – Invoca a função ***Send_And_Receive_Frame*** e verifica se o envio e receção das tramas foi realizado corretamente. Na ausência de erros, é responsável pelo envio da trama UA.

b. **Send_And_Receive_Frame -** Envio da trama DISC (Disconect) e aguarda receção da trama DISC.

c. **Check_State_Machine –** Verifica se a trama recebida é a pretendida (DISC).

Sequência de funções no recetor

1. llopen - Estabelece a ligação através da porta de série:

a. *Set_Serial_Port* - Configura a porta de série;

b. *Open_Receiver* – Aquando da receção da trama SET, envia aa trama UA (Unnumbered Acknowledgment) para o emissor;

c. Check_State_Machine – Verifica se a trama recebida é a pretendida (SET).

2. llread - Receção da trama de informação: se existirem erros no cabeçalho a trama é descartada. Faz *destuffing* da trama e sempre que existirem erros no campo de dados, nomeadamente, no campo de controlo (BCC2), é enviada uma trama de supervisão REJ de forma a pedir uma retransmissão dos dados. Na ausência de erros, ou na presença de uma réplica, uma trama de supervisão RR é enviada;

a. Receive_Data – Verifica falhas na trama e envia uma trama RR na ausência de erros;

b. Check_State_Machine – Verifica se a o cabeçalho da trama I está correto.

***3. llclose* -** Encerra a ligação;

a. *Close_Receiver* – Aquando da receção da trama DISC, envia uma trama DISC;

b. Check_State_Machine – Verifica se a trama recebida é a trama pretendida (DISC).

Protocolos de ligação lógica

A camada de ligação de dados é a camada de mais baixo nível e é a camada que interage diretamente com a Porta Série. Deste modo, destacam-se os principais **aspetos funcionais**:

- Estabelecimento e fecho da ligação;
- Envio e receção de informação através da porta de série;
- Controlo de erros – Stuffing e Destuffing / Numeração das tramas / Valor do campo de dados (BCC2);

Estabelecimento e fecho da ligação

A função llopen estabelece o início da ligação entre os dois computadores através da porta de série. Depois da configuração da porta de série, no lado do emissor, é chamada a função *Open_Transmitter*, que vai enviar uma trama *SET* e aguardar pela receção de um *UA_R*. No lado do recetor é invocada a função *Open_Receiver*, que envia a trama *UA_R* após a chegada da trama *SET*.

A função llclose garante o fecho da ligação. Do lado do transmissor, é invocada a função *Close_Transmitter* que irá enviar uma trama do tipo DISC. Após o envio, o transmissor aguarda a

mesma trama, desta vez, proveniente do recetor. Finalmente, depois da confirmação da trama *DISC*, é enviado uma trama *UA*. Inversamente, no lado do recetor, é invocada a função *Close_Receiver* que irá aguardar a receção da trama *DISC*. Por conseguinte, a mesma trama é enviada e, o fecho da ligação é realizado no momento da receção da trama *UA* proveniente do emissor.

Envio e receção de informação através da porta de série

A função llwrite é responsável por, principalmente, encapsular a trama proveniente da camada de aplicação e fazer o *stuffing* dos dados. Após este processo, a trama é enviada para a porta série invocando a função *Send_And_Receive_Frame*. Como todas as tramas enviadas pelo transmissor exigem uma confirmação por parte do recetor (Protocolo Stop & Wait), esta função permite enviar uma trama e aguardar pela resposta pretendida. É ainda possível, juntamente com a função Check*_State_Machine*, a identificação de tramas de confirmação negativa, possibilitando uma retransmissão sempre que necessário.

No lado do recetor, e uma vez que a camada de aplicação apenas necessita do campo de dados, a função llread é responsável pela realização do processo inverso. Para isso, é chamada a função *Receive_Data*. Esta é responsável pelo desencapsulamento da trama e a respetiva recuperação dos dados originais ao realizar o *destuffing*. É ainda nesta função que se realiza a verificação de erros no cabeçalho e no campo de dados da trama, retransmitindo a mesma se necessário.

Controlo de erros

- **Stuffing e destuffing das tramas**

O Stuffing consiste na inserção de bits não informativos no campo de dados. Neste caso, os bits acrescentados têm como objetivo impedir que o recetor termine a leitura do campo de dados prematuramente, após encontrar uma FLAG inserida no mesmo.

Assim, antes do transmissor enviar a trama encapsulada, verifica bit a bit a existência de uma FLAG no campo de dados. Em caso afirmativo, substitui esse bit por um ESC e na próxima posição do vetor guarda o resultado da operação XOR entre a FLAG e 0x20. No lado do recetor, sempre que é detetado um ESC, é realizado o processo inverso, o destuffing.

- **Numeração de tramas através do número de sequência**

Sempre que o transmissor envia uma trama, esta tem um número de sequência associado que alterna entre 0 e 1 a cada trama confirmada. Se por algum motivo, o recetor receber uma trama correta e a respetiva confirmação não chegar ao transmissor, este, após o *timeout*, irá enviar a mesma trama. Através do número de sequência, o recetor tem a capacidade de identificar que esta trama é uma réplica, enviando uma trama do tipo RR, mas descartando o campo de dados.

- **Valor do campo de dados (BCC2)**

Para o recetor certificar-se que o campo de dados está ausente de erros, utiliza o byte BCC2. Primeiramente, no lado do transmissor, o BCC2 é calculado (XOR recursivo desde D1 a DN, sendo D cada bit do campo de dados) e enviado na trama encapsulada. No lado do recetor, o BCC2 é calculado localmente e comparado com o enviado pelo emissor. Caso sejam iguais, então o campo de dados chegou corretamente ao recetor.

Validação

De forma a analisar a eficiência do protocolo, foram realizados os seguintes testes:

- Variação do tamanho das tramas do tipo I – Baudrate - 139000 / Tamanho do ficheiro – 475,3Kbytes

O primeiro teste realizado consiste na variação do payload máximo da trama e na consequente alteração do tamanho das tramas de informação. Como é evidenciado na figura 1, quanto maior o tamanho da trama de informação, menos tramas serão enviadas e a informação é transmitida mais rapidamente, aumentando assim a eficiência. No entanto, em condições mais propícias ao erro, a trama necessita de uma retransmissão e, para baudrates mais baixos, pode aumentar o tempo de processamento e diminuir a eficiência.

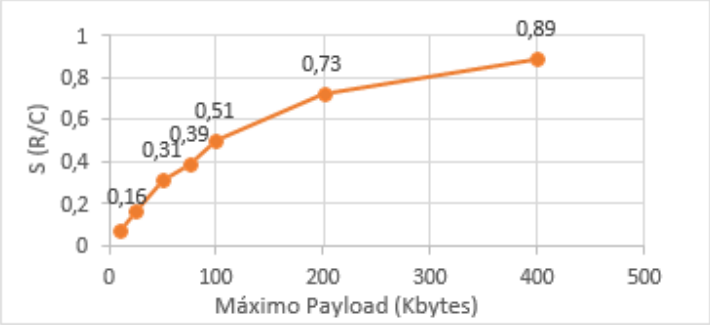

Figura 1. Variação do tamanho das tramas do tipo I

- Variação do Baudrate – Tamanho do ficheiro 11Kbytes

Após os testes realizados, foi concluído que o baudrate não altera significativamente a eficiência do protocolo. No entanto, para valores mais elevados, o tempo de processamento diminui de forma excessiva. Esta diminuição deixa de alterar a taxa de transmissão, diminuindo assim a eficiência.

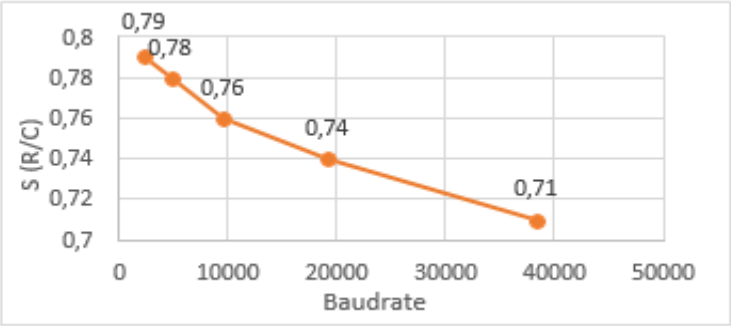

Figura 2. Variação do Baudrate

- Variação do FER

Após a análise do gráfico da figura 3, conclui-se que o aumento da percentagem de erros em BCC1 e BCC2 tem um impacto negativo no tempo de transferência das tramas. No entanto, o erro em BCC1 é mais prejudicial uma vez que, neste tipo de erros, a trama é descartada e o transmissor aguarda um timeout, reduzindo assim a eficiência do protocol.

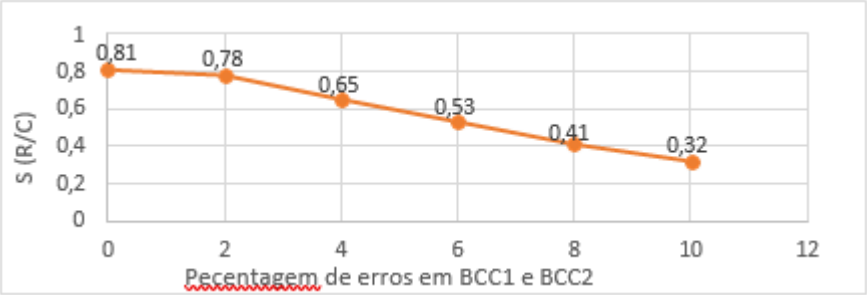

Figura 3. Variação do FER

Conclusões

O trabalho realizado permitiu a realização de um protocolo de ligação de dados, que estabelece uma conexão assíncrono, por meio de uma porta de série, entre dois computadores. A abordagem direta com estes conceitos, se por um lado, forneceu uma aprendizagem mais assertiva relativamente à camada protocolar, por outro, permitiu percecionar a elevada complexidade deste tipo de sistemas de comunicação.

A principal dificuldade encontrada na realização do trabalho recaiu sobre a necessidade de adaptar o protocolo realizado, que por si só já complexo e desafiador, à camada de aplicação.

Em suma, e com os testes efetuados, o nosso grupo conclui que o protocolo, com os parâmetros adequados, atinge uma elevada eficiência e resiliência a erros cumprindo assim com os objetivos pretendidos.