Faculdade de Engenharia da Universidade do Porto

1º Trabalho Laboratorial

Rede de Computadores

2017/2018 -- Mestrado Integrado em Engenharia Informática e Computação:

Turma 3:

Afonso Bernardino da Silva Pinto

(up201503316@fe.up.pt)

Filipe Miguel Leitao Ribeiro

(ei11141@fe.up.pt)

Tomás Sousa Oliveira

(up201504746@fe.up.pt)

Docentes:

Manuel Ricardo mricardo@fe.up.pt Maria Teresa Andrade mandrade@fe.up.pt

Sumário

Criado no âmbito da disciplina Rede de Computadores leccionada no 3º ano do Mestrado Integrado em Engenharia Informática e Computação, este trabalho tem como objetivo aplicar os conceitos sobre protocolos de transferência de dados, no qual, consiste no envio de dados entre dois computadores, usando uma porta de série.

Palavras-Chave

Emissor; Recetor; Transferência de Dados; Porta de Série; C; Rede de Computadores; FEUP.

Agradecimentos

Este projeto foi resultado de diversas contribuições e colaborações, dada de forma direta e indireta, mas todas elas essenciais à sua realização. Gostaríamos assim de expressar os nossos sinceros agradecimentos a todos os que tornaram possível este trabalho, especialmente à professora Maria Teresa Andrade pela orientação dada e valioso acompanhamento constante durante o desenvolvimento do trabalho.

Índice

1. Introdução

Este projeto foi realizado no âmbito da unidade curricular Redes de Computadores, do 3º ano do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto.

O objetivo deste projeto é implementar um protocolo para transferência de dados através de uma porta de série RS-232.

Para realizar este processo, necessitamos de implementar protocolos de encapsulamento e usar mecanismos de deteção e controlo de erros, de modo a que a informação que o computador recetor receber seja seguramente correta.

O presente relatório servirá para explorar as funcionalidades deste protocolo e fornecer detalhes sobre a sua implementação.

Para uma melhor leitura do mesmo optamos por dividir o relatório nas seguintes secções:

- Arquitetura - Demonstra os blocos funcionais e explica a interface com o utilizador;
- Estrutura do código - Apresenta as funções e estruturas de dados principais de cada camada;
- Casos de Uso Principais - Principais funcionalidades do programa;
- Protocolo de ligação lógica - Identifica os principais aspetos funcionais e explica, com excertos do código, a estratégia implementada;
- Protocolo de aplicação - Identifica os principais aspetos funcionais e explica, com excertos do código, a estratégia implementada;
- Validação - Descreve alguns testes efetuados e respetivos resultados;
- Eficiência do protocolo de ligação de dados - Demonstra a estatística da eficiência do protocolo, com recurso a medidas do código.

2. Arquitetura

O nosso projeto está dividido em 2 camadas:

- a camada de ligação de dados - contém as funções responsáveis pelo estabelecimento e término corretos da ligação, bem como funções genéricas de implementação do protocolo (sincronismo, controlo de erros, stuffing, entre outros).

- a camada de aplicação - depende diretamente da camada de ligação de dados, delegando-lhe grande parte do seu trabalho. É o responsável pela transferência de ficheiros, gerindo-a através da emissão e receção de tramas.

A nível de interface com o utilizador o nosso programa oferece a oportunidade de escolher os parâmetros (port, baud rate, número de transmissões, time-out, tamanho da trama, induzir a probabilidade de erros e aumentar tempo de propagação) que quiser (dentro de uma basta gama de hipóteses)

3. Estrutura do código

Cada excerto de código será explicado nos tópicos 5 e 6 sobre a camada de ligação de dados e a camada de aplicação, respetivamente.

3.1 Camada de ligação de dados

As principais funções utilizadas por esta camada:

```
int llopen(LinkLayer* linkLayer);
int llwrite(LinkLayer* linkLayer, unsigned char* buffer, int bufferSize);
int llread(LinkLayer* linkLayer);
int llclose(LinkLayer* linkLayer);
```

Estrutura da Link Layer:

```
typedef struct {
  /*Connection Data */
  int fileDescriptor;
  Mode mode; /*TRANSMITTER | RECEIVER*/
  char* port;
  int baudRate; /*Transmission Speed*/
  unsigned int timeout; /*Timer Value*/
  unsigned int numTransmissions;  /*Num of tries in case of error*/
  unsigned int dataSize;

  /*Connection Helpers*/
  unsigned int sequenceNumber;

  /*FIle Data*/
  char* fileName;
  unsigned int fileSize;

  /* Statistics */
    unsigned int numSentRR;
    unsigned int numReceivedRR;
    unsigned int numSentREJ;
    unsigned int numReceivedREJ;
  unsigned int numTimeouts;
  double timeElapsed;

  /*Error Induction*/
  unsigned int induceError;
  unsigned int increaseTProg;

  //frame
   unsigned char* frame;
} LinkLayer;
```

3.2 Camada de aplicação

As principais funções utilizadas por esta camada:

```
int appLayer(LinkLayer* linkLayer);
int sendData(LinkLayer* linkLayer);
int receiveData(LinkLayer* linkLayer);
int sendControlPackage(LinkLayer* linkLayer, ControlPacket* controlPacket);
int sendDataPackage(char* buffer, int N, int length, LinkLayer* linkLayer);
int receiveControlPackage(int* controlPackageType, LinkLayer* linkLayer);
int receiveDataPackage(int* N, char** buf, int* length, LinkLayer* linkLayer);
```

Estruturas Auxiliares:

```
typedef struct {
    ControlPacketType type;
    unsigned char length;
    char* value;
} ControlPacketTLV;

typedef struct {
    PacketControlField controlField;
    unsigned int numParams;
    ControlPacketTLV *params;
} ControlPacket;

typedef struct {
    PacketControlField controlField;
    unsigned char sequenceNumber;
    unsigned int length;
    char* dataBuffer;
} DataPacket;
```

4. Casos de uso principais

Este projeto é ultimamente utilizado para a transferência de um qualquer ficheiro entre 2 computadores ligados entre si através de uma porta de série.
Para executar a aplicação basta correr o programa em ambos os computadores e configurar a ligação conforme solicitado (parâmetros da ligação, nome do ficheiro, indução de erros).

5. Protocolo de ligação lógica

A *Link Layer* é uma das camadas implementadas neste projeto, e tem várias funcionalidades, entre as quais se destacam:

- Estabelecimento e terminação de uma ligação através da porta de série;
- Escrita e leitura de mensagens da porta de série;
- Criação e envio de comandos e mensagens através da porta de série;
- Receção de mensagens através da mesma:
- Receção dos packets da *Application Layer* e posterior *Stuffing* e *Destuffing.*

As funções de maior importância que foram implementadas nesta camada são *llopen*, *llwrite*, *llread* e *llclose* e estão a seguir descritas.

5.1 Função *llopen()*

Esta função é responsável por estabelecer uma ligação entre dois computadores através da porta de série.
Primeiro, o emissor, ao invocar esta função, envia o comando *SET* e aguarda que o recetor responda, sendo a resposta o comando *UA.* No entanto, é possível que esta resposta não seja enviada ou não chegue, havendo um alarme, com tempo de *timeout,* que assegura que o emissor não fica indefinidamente à espera de uma resposta que poderá nunca chegar. Neste caso, o emissor volta a enviar o comando *SET* por um número de tentativas determinado. Após se esgotarem as tentativas (3, por predefinição), a ligação que havia sido efetuada, é então terminada.
Do outro lado, quando esta função é invocada, o recetor fica à espera de receber o comando *SET* e posteriormente envia o comando *UA,* estabelecendo-se então a ligação.

5.2 Função *llwrite()*

Esta é a função usada pelo emissor para enviar os dados para o recetor, recorrendo a funções auxiliares como *sendData()* da camada da aplicação, aguardando, tal como na função anteriormente descrita, uma resposta por parte do recetor.
Nesta função, também existe um alarme com número máximo de tentativas. Se não for recebida resposta, os dados voltam a ser enviados.
Se houver resposta e esta for o comando *RR*, então os dados enviados foram transferidos com sucesso. Caso a resposta seja o comando *REJ*, então os dados não foram transferidos com sucesso, o que faz com que seja efetuada uma nova tentativa e o número de tentativas assumidas pela função volta a zero.

5.3 Função *llread()*

Esta função é responsável por receber e ler a mensagem enviada pelo emissor. Se esta mensagem for inválida, então envia o comando *REJ*, para que esta seja enviada novamente. Se a mensagem for válida, e for uma mensagem de informação, então o

comando RR é enviado e a informação recebida é guardada no computador do recetor. No entanto, se receber o comando *DISC,* a ligação será terminada, chamando a função *llclose().*

5.4 Função *llclose()*

Esta função é responsável por terminar a ligação anteriormente efetuada entre os dois computadores através da porta de série. Quando esta função é chamada pelo emissor, o comando *DISC* é enviado, e é aguardada a receção de um comando *DISC*. Quando isto se verifica, o comando *UA* é enviado, e a ligação é então terminada.

6. Protocolo de aplicação

A *Application Layer* é a camada de mais alto nível que foi implementada e tem como funcionalidades:

- Envio e receção de *control packets;*
- Envio e receção de *data packets;*
- Envio e receção do ficheiro pretendido.

Tal como descrito, os *packets* na camada da aplicação podem ser *packets* de controlo (inicial ou final) ou *packets* de *data* (informação). Estes podem ser diferenciados através do seu primeiro *byte.* Quando este assume o valor 1, significa que o *packet* é de informação (*data packet*). Se assumir o valor 2, o *packet* é de controlo inicial e caso assuma o valor 3, é um *packet* de controlo final.
As funções de maior destaque nesta camada são *sendData(), receiveData(), sendDataPackage(), receiveDataPackage()*, *sendControlPackage()* e *receiveControlPackage()* e estão a seguir descritas:

6.1 Função *sendData()*

Esta função serve para que o emissorpossa enviar os *control packets,* e abrir o ficheiro definido, ler o seu conteúdo e enviar o mesmo em várias tramas. Em primeiro lugar, chama a função *sendControlPackage()* para que possa enviar o pacote de controlo *start.* Depois, chamando a função *sendDataPackage()* envia a informação do ficheiro, *packet* a *packet* e finalmente, envia o pacote de controlo *end,* voltando a chamar a função *sendControlPackage().*

6.2 Função *receiveData()*

Esta função serve para que o recetor possa ler as tramas que foram enviadas pelo emissor. Recorre às funções *receiveDataPackage()* e *receiveControlPackage()* para processar os *packets* recebidos. A primeira é chamada para processar os *data packets* e a segunda para verificar os *control packets start* e *end.*

6.3 Função *sendDataPackage()*

Esta função permite ao emissor formar a trama de dados de uma forma conveniente.

6.4 Função *receiveDataPackage()*

Esta função é chamada para processar os *data packets* recebidos pelo recetor. Esta, caso a leitura destes seja efetuada com sucesso, escreve a sua informação no computador, enviando o comando *RR*. Caso esta leitura não seja efetuada com sucesso, é enviado o comando *REJ* e esta informação terá de ser enviada novamente.

6.5 Função *sendControlPackage()*

Esta função serve que o emissor, antes de enviar os *data packets* ao recetor, envie *packets* de controlo, para que o recetor saiba quando começa e termina o envio da informação do ficheiro definido. Esta função é responsável também por enviar a informação (nome e tamanho) do ficheiro.

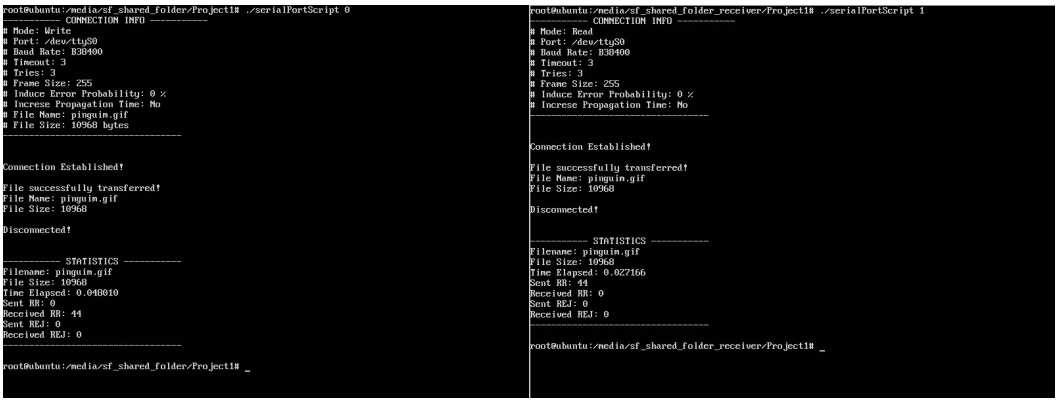
6.6 Função *receiveControlPackage()*

Esta função é chamada para processar os *control packets* recebidos pelo recetor. Verifica se o primeiro packet recebido é *control packet start* e se o último é *end,* terminando então a receção do ficheiro.

7. Validação

De modo a validar a nossa implementação, foram realizados vários testes[1] variando as várias componentes de ligação (baudrate, tamanho do packet, número de tentativas, segundos de espera). Todas as transferências foram concluídas com sucesso.

Posteriormente, esses testes passaram a incluir a introdução de erros (tanto por interrupção do sinal como por introdução de “lixo”). Nestas condições o programa conseguia detetar a ocorrência de erros mas revelou algumas falhas em retomar o envio de alguns ficheiros.



Output do programa referente à transferência de pinguim.gif

[1] Alguns deles encontram-se em anexo

8. Eficiência do protocolo de ligação de dados

		Avg Time (µs) - Tf(practical)			S (practical)		S (theoritical)	St/Sp		Tp(µs)
Baudrate	packet_size (bytes)	Transmitter	Receiver	Tf (theoritical)	Transmitter	Receiver		Transmitter	Receiver	0,01
19200	128	92413	30077	53333,33333	0,999999784	0,999999335	0,999999625	0,999999841	1,00000029	
19200	256	52710	25315	106666,6667	0,999999621	0,99999921	0,999999813	1,000000192	1,000000603	
19200	1024	14506	29893	426666,6667	0,999998621	0,999999331	0,999999953	1,000001332	1,000000622	
38400	128	49162	40035	26666,66667	0,999999593	0,9999995	0,99999925	0,999999657	0,99999975	
38400	256	48980	25295	53333,33333	0,999999592	0,999999209	0,999999625	1,000000033	1,000000416	
38400	1024	13231	17333	213333,3333	0,999998488	0,999998846	0,999999906	1,000001418	1,00000106	
57600	128	85196	32200	17777,77778	0,999999765	0,999999379	0,999998875	0,99999911	0,999999496	
57600	256	47590	27934	35555,55556	0,99999958	0,999999284	0,999999438	0,999999858	1,000000153	
57600	1024	10140	17580	142222,2222	0,999998028	0,999998862	0,999999859	1,000001832	1,000000997	

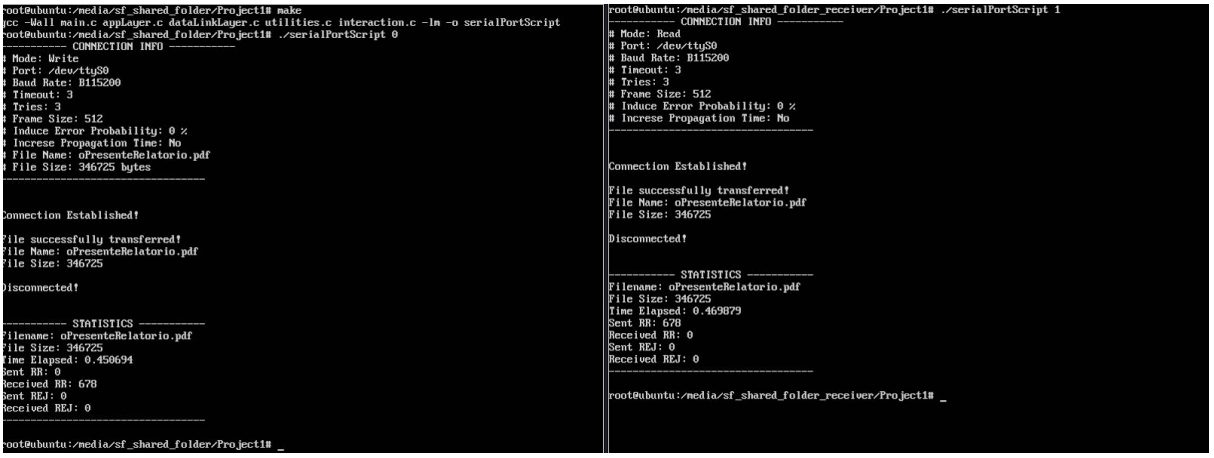
9. Conclusão

Com a realização deste projeto pudemos familiarizar-mos-nos com a implementação de protocolos e a utilização de porta-séries.

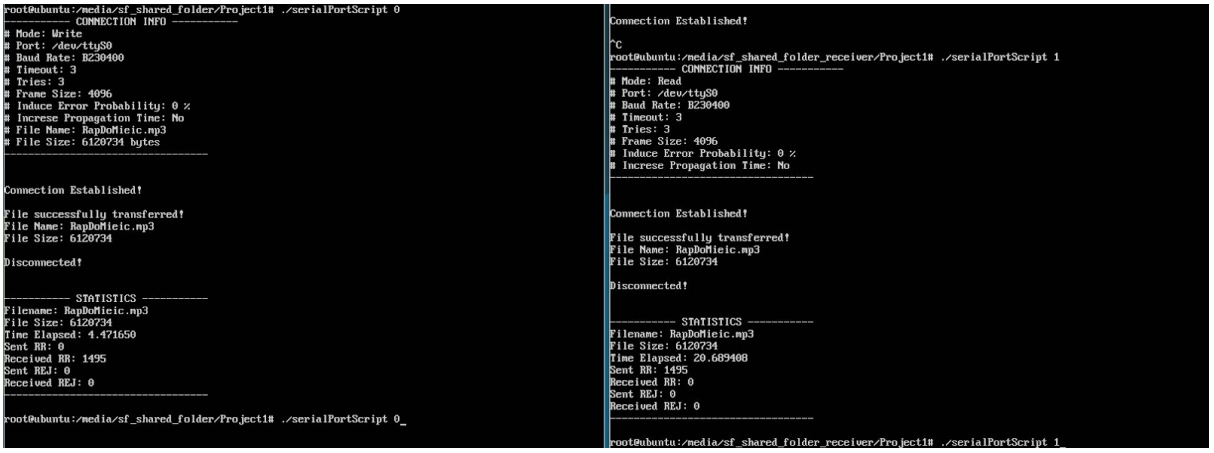
Utilizando técnicas de sincronização, conservação da transparência e controlo de erros, criamos um programa estruturado em camadas independentes entre si capaz de efectuar a transmissão de diversos tipos de ficheiros através da porta de série RS-232.

Anexos

Anexo I - Testes de validação



Output do programa referente à transferência de oPresenteRelatorio.pdf



Output do programa referente à transferência de RapDoMieic.mp3