22 de dezembro de 2021

**Projeto de Redes de Computadores**

**Aplicação de Transferência de Ficheiros com Porta Série**

**Autores**

Mateus Silva, up201906232

Melissa Silva, up201905076

**Sumário**

O presente relatório aborda o funcionamento do primeiro projeto realizado para a unidade curricular de *Redes de Computadores*, onde se pretendia a elaboração de uma aplicação capaz de transferir ficheiros entre dois computadores com diferentes papéis (*recetor* e *transmissor*) ligados por uma porta série. A aplicação deveria seguir um modelo de camadas: o maior nível, com a aplicação em si e a sua interação com o utilizador (que lhe fornece argumentos); o menor nível, que efetue a transmissão de dados do ficheiro e finalmente a camada de ligação entre as duas últimas.

Após vários testes realizados através de portas série virtuais e nas portas séries físicas fornecidas em laboratório, concluímos que o trabalho foi realizado com sucesso e que cumprimos todos os objetivos estabelecidos no guião. O resultado é então uma aplicação completamente funcional capaz de transmitir diferentes tipos de ficheiros (imagens, documentos, texto, áudio e vídeo) entre dois computadores sem perda de dados.

**Introdução**

O projeto tinha como objetivos a elaboração de um mecanismo de transmissão de dados por camadas, em que uma destas intermediasse os dois níveis mais extremos, sendo estes os da aplicação e o *driver* da porta série.

O protocolo da camada intermédia deveria possuir um mecanismo de transparência por *byte stuffing* e criação de tramas de três possíveis tipos com componentes estruturais comuns (cabeçalho) mas que se diferenciam nas demais devido ao papel têm. Tramas de supervisão e não numeradas possuem proteção simples, enquanto tramas de informação possuem proteção dupla. A respeito da transmissão de tramas (*driver* da porta série), deveria usar-se um mecanismo de *Stop and Wait* com *timeout* e tentativa de retransmissão de tramas. Cada interveniente na transmissão (*emissor* e *recetor*) recebe e envia tramas; contudo, apenas o emissor pode transmitir pacotes – e, portanto, apenas ele pode transmitir tramas de informação. A aplicação deveria suportar e interpretar dois tipos de pacotes: de controlo e de dados. Os primeiros servem para sinalizar o início e fim de uma transmissão, tendo uma estrutura diferente.

Finalmente, espera-se que haja independência entre todas as camadas, pelo que a camada intermédia (ligação de dados) não pode processar cabeçalhos dos pacotes; a camada de aplicação não pode conhecer os detalhes da camada intermédia, apenas como lhe pode aceder; finalmente, a numeração de tramas de informação e pacotes de dados devem ser totalmente independentes.

As secções 5 e 6 detalham o funcionamento de cada uma das camadas do projeto.

**Arquitetura**

O projeto possui duas camadas independentes: aplicação, ligação de dados. A primeira encontra-se totalmente nos ficheiros app, tratando de leitura e criação de pacotes de dados e pacotes de controlo. A camada de ligação de dados encontra-se em *protocol\_app* (*llopen(), llclose()*, *llwrite()*, *llread()* e criação de tramas de informação), stuffing (para o mecanismo de transparência) e em data\_protocol (leitura e escrita de tramas de informação e comandos).

O utilizador chama um dos resultados da compilação dos ficheiros de código – *receiver* e *transmitter* *–* para iniciar o programa como recetor ou emissor, respetivamente.

**Estrutura do Código**

O código tem duas vertentes, uma para cada interveniente da transmissão. Isto resulta em dois executáveis, *receiver* e *transmitter*, cuja estrutura exploramos a seguir.

**Transmitter *(Emissor)***

A interação com este executável faz-se usando comandos com o seguinte formato no terminal.

./transmitter [porta\_série] (-t timeout) (-n tent\_falha) [-f nome do ficheiro]

[porta\_série] é um argumento obrigatório para a definição da ligação com o transmissor através da porta série.

(-t timeout) permite indicar um *timeout* em segundos usado no mecanismo de *Stop and Wait*. Se não indicado, existe um valor *default* de 3 segundos.

(-n tent\_falha) permite indicar um número máximo de tentativas no caso de ocorrerem erros na transmissão. Se não indicado, existe um valor *default* de 3 tentativas.

[-f nome\_ficheiro] é um argumento obrigatório para indicar o ficheiro a transmitir.

Da camada de ligação de dados, chama:

llopen – inicia o programa, abrindo a comunicação através das entradas na porta série, envia uma trama de supervisão SET e espera por uma trama não numerada UA de *receiver*.

a

llwrite – escreve as várias tramas de informação e está preparado para tentar retransmiti-las caso a informação não chegue corretamente a *receiver*. Precede-se a escrita com um processo de *byte stuffing* aplicado às tramas.

**I**

llclose – cessa a comunicação entre as entradas da porta série. Isto é precedido pelo envio de uma trama DISC a *receiver*, esperando-se por uma trama igual deste, após o que se envia uma trama UA.

Da camada da aplicação, chama:

create\_control\_package – cria um pacote de controlo, quer este seja de início ou fim de transmissão, com uma componente em formato TLV. A informação para um pacote provém do ficheiro fornecido a *transmitter*.

**I**

read\_data\_package – cria um pacote de dados, com informação proveniente do ficheiro fornecido a *transmitter*.

**Receiver *(Recetor)***

Invoca-se o executável com comandos do seguinte formato.

receiver [porta\_série] [-f nome\_ficheiro]

[porta\_série] é um argumento obrigatório para a definição da ligação com o transmissor através da porta série.

[-f nome\_ficheiro] é um argumento obrigatório para definir um novo nome para o ficheiro recebido.

Da camada de ligação de dados, chama:

o

llopen – inicia o programa, abrindo a comunicação através das entradas na porta série, recebe uma trama de supervisão SET e envia uma trama não numerada UA.

llread – lê tramas de informação enviadas por *transmitter*, efetua *byte destuffing* e valida a informação antes de a guardar. De outra forma, envia uma trama REJ, solicitando uma retransmissão.

Da camada da aplicação, chama:

llclose – cessa a comunicação através das entradas da porta série. Espera por uma trama DISC de *transmitter*, envia uma de volta a este e espera por uma trama UA para confirmar tal ação.

oi

read\_data\_package – lê e guarda a informação contida num pacote de dados.

read\_control\_package – lê e guarda a informação contida na componente de formato TLV do pacote de controlo, quer este seja de início ou de fim de transmissão.

**Estruturas de Dados**

struct linkLayer {

char port[20]; // Dispositivo /dev/ttySx, x = 0, 1

int baudRate; // Velocidade de transmissão

unsigned int sequenceNumber; // Nº de sequência da trama (0,1)

unsigned int timeout; // Valor do temporizador: 1s

unsigned int numTransmissions; // Tentativas em caso de falha

int fileDescriptor; // Descritor da porta série

int status; // TRANSMITTER | RECEIVER

};

struct applicationLayer {

int sequenceNumber; // Nº de sequência do pacote de dados esperado

};

typedef enum {

START,

FLAG\_RCV,

A\_RCV,

C\_RCV,

BCC\_OK,

D\_RCV,

STOP

} messageState

**Casos de Uso Principais**

Existem dois casos de uso principais: receção do ficheiro (*receiver*) ou envio do ficheiro (*transmitter*). Mostramos a seguir a sequência de chamadas das funções principais mencionadas na secção anterior.

*transmitter*

1. llopen()
2. create\_control\_package();
3. llwrite();
4. Ciclo:
   1. create\_data\_package();
   2. llwrite();
5. create\_control\_package();
6. llwrite();
7. llclose();

*receiver*

1. llopen()
2. llread()
3. read\_control\_package()
4. Ciclo
   1. llread()
   2. read\_data\_package();
5. llread();
6. read\_control\_package();
7. llclose();

**Protocolo de Ligação Lógica**

**llopen() – Estabelecimento da Ligação Lógica**

Se for *transmitter* a chamar esta função, abre-se a porta série e envia-se uma trama SET para *receiver*, usando a função send\_cmd(). Espera-se por uma trama UA como resposta de *receiver*, e esta será lida usando read\_cmd(), que implementa a máquina de estados do **Anexo I**. Ativa-se também o mecanismo de alarme e, caso ocorra *timeout*, ocorrerão até *n* tentativas de retransmissão. Após a *n-*ésimatentativa falhada, o programa termina.

Se for *receiver* que chama esta função, abre-se a porta série, lê-se a trama SET recebida com read\_cmd()e envia-se uma trama UA em resposta, usando send\_cmd().

**llwrite() – Escrita de Tramas de Informação**

Apenas *transmitter* chama esta aplicação. Primeiro, assegura-se que a informação a transmitir é guardada caso seja necessário retransmiti-la. Segue-se a criação da trama de informação com o formato adequado usando a função create\_info\_trama(). Esta última cria o componente BCC2 através de uma operação XOR de todos os *bits* presentes na componente dos dados, e ambas as componentes sofrerão um processo de *byte stuffing*. A trama é enviada ao *receiver* e espera-se a sua resposta. Pelo mecanismo de alarme, caso não recebamos uma resposta antes de terminar um período de *timeout*, retransmissão irá ocorrer.

Assumindo que não há *timeout*, verificamos a resposta de receiver: se for RR, então a informação enviada foi aceite, pelo que validamos o número de sequência esperado (que deve ser oposto ao atual em llwrite(), guardado numa variável estática para este propósito). Se for válido, desativamos o alarme, mudamos o valor de sequência de llwrite() e sai-se da função. Se não for válido, vamos retransmitir a informação até obtermos a resposta esperada ou excedermos o número de tentativas máximo. De outra forma (resposta REJ), desativamos o alarme e procederemos a retransmitir a informação até obtermos a resposta esperada.

**llread() - Leitura de Tramas de Informação**

Apenas *receiver* chama esta função. Primeiro, usa-se read\_info\_trama() para efetuar a leitura da trama de informação recebida de *transmitter.* A função mencionada implementa a máquina de estados do **Anexo 2**. Depois, sujeita-se a trama a *destuffing*. O resultado é usado para obter BCC2, efetuando uma operação XOR entre todos os bits da trama (exceto o último, já que esse é o verdadeiro valor de BCC2, para verificação).

Verifica-se primeiro se a informação recebida é duplicada, através de uma validação do número de sequência recebido com o valor esperado. Se for inválido, a informação é repetida e pediremos retransmissão enviando uma trama RR de número de sequência oposto ao recebido, descartando a informação recebida. De outra forma, efetuamos a validação entre os valores de BCC2 mencionados: se forem iguais, a informação chegou corretamente, pelo que a guardamos, enviamos para *transmitter* uma trama RR e mudamos o número de sequência esperado, saindo, depois, da função. Caso contrário, enviamos uma trama REJ como pedido de retransmissão.

**llclose()** - **Fecho da Ligação Lógica**

Se for *transmitter* a chamar esta função, enviamos uma trama DISC a *receiver*, usando a função send\_cmd(). Espera-se a resposta de *receiver*, que deverá ser uma trama DISC, que será lida usando read\_cmd() (**Anexo I**). Caso não haja uma resposta antes de ocorrer *timeout*, então o alarme será ativado e haverá, no máximo, *n* tentativas de retransmissão. Caso isto não surta sucesso, o programa termina por aqui. De outra forma, envia-se uma trama UA e procede-se ao fecho da porta série.

Se for *receiver* a chamar esta função, irá ler-se a trama DISC recebida através de read\_cmd(), enviar-se-á uma trama DISC em resposta usando send\_cmd() e ler-se-á a trama UA recebida, novamente com read\_cmd(), e, por fim, fecha-se a porta série.

**Protocolo da Aplicação**

**create\_control\_package()**

Esta função cria os dois tipos de pacotes de controlo para o início e fim de uma transmissão. É apenas chamada por *transmitter* e recebe deste a informação do ficheiro a transmitir, que é expressa no formato TLV (tipo, tamanho, valor) no pacote. A informação transmitida é o tamanho e o nome do ficheiro. Pacotes de controlo são expressos em *arrays* de *u\_int8\_t*.

**create\_data\_package()**

Chamada apenas por transmitter, recebendo como parâmetros o número do pacote a criar, o seu tamanho e uma porção do conteúdo do ficheiro. É feito o processamento destes dados e depois faz-se o pacote com o formato desejado.

Pacotes de dados são expressos em *arrays* de *u\_int8\_t*.

**read\_control\_package()**

É chamada por *receiver*. Lê pacotes de controlo e guarda a informação encontrada nos campos valor das componentes TLV destes.

**read\_data\_package()**

Apenas chamada apenas por *receiver*, esta função lê pacotes de dados. Para tal, começa por verificar se está a lidar com um pacote repetido (validando o número de sequência de pacote com o número de sequência esperado). Se verificar que um pacote é repetido, o programa é terminado pois houve perdas de informação. De outra forma, extrai-se a informação do pacote.

**Validação**

Testámos o nosso código em múltiplos cenários.

**1.** Transmissão da imagem sugerida: pinguim.gif;

**2.** Transmissão de imagens de tamanho maior que a sugerida;[[1]](#footnote-1)

**3.** Transmissão de ficheiros de outros formatos;[[2]](#footnote-2)

**4.** Interrupção da Transmissão (desconetando/reconectando a porta série);

**5.** Transmissão com Indução de Erros (aproximando barra de ferro à porta série);

**6.** Transmissão com Variação dos valores de *Baudrate;*

**7.** Transmissão com Variação dos valores em tramas de informação;

**8.** Transmissão com Variação dos valores de *T\_prop;*

**9.** Transmissão com Variação de FER em Percentagem.

O nosso código passou em todos os testes com sucesso.

**Secção 8**

**Conclusão**

Durante o período de trabalho dedicado a este projeto, conseguimos implementar tudo aquilo que era esperado com sucesso, resultando numa aplicação completamente funcional e subsistente para a transferência de ficheiros entre dois computadores através de uma porta série de uma forma segura.

No geral, acreditamos que o projeto é de uma complexidade elevada, embora tenhamos conseguido atingir os seus objetivos. A parte onde encontrámos mais dificuldade foi para a obtenção de dados para o estudo estatístico feito na secção anterior, visto que só poderíamos obter medições em laboratório. Mesmo assim, essas dificuldades foram ultrapassadas e gostaríamos de demonstrar orgulho em ter completado este projeto.

**Anexo I –** Diagrama da Máquina de Estados Implementada em read\_cmd()

Diagram, schematic

Description automatically generated

**Anexo II –** Diagrama da Máquina de Estados Implementada em read\_info\_trama()

A picture containing text, device

Description automatically generated

1. 92KB, 200MB, 602MB, 1.8MB, 3.3MB, 7.8MB. [↑](#footnote-ref-1)
2. MP3, JPG, PNG, MP4. [↑](#footnote-ref-2)