1º Trabalho Laboratorial - Ligação de Dados

FEUP - RC - Turma 7, Grupo 7

2021/22

Duarte Guedes Sardão	José Pedro Ferreira	Lucas Calvet Santos
up201905497	up201904515	up201904517

Sumário

Este trabalho laboratorial foi desenvolvido no âmbito da Unidade Curricular de Redes de Computadores e teve como objetivo aplicar na prática os conceitos teóricos lecionados na mesma, mais precisamente implementar uma aplicação e protocolo que permitissem a transmissão de ficheiros através de uma porta série assíncrona, protegida de possíveis erros de transmissão.

Os objetivos do trabalho, estabelecidos no guião, foram cumpridos com sucesso. Foi desenvolvida uma aplicação funcional, capaz de lidar com qualquer tipo de ficheiro e transmiti-lo entre dois computadores, sem perda de dados.

Introdução

O trabalho teve como objetivo desenvolver uma aplicação, suportada por um protocolo de dados, que, com recurso à comunicação por tramas de informação e através de uma porta série, tem a capacidade de funcionar como um mecanismo de transferência de ficheiros entre computadores, resiliente à ocorrência de falhas de conexão.

Este relatório serve como complemento ao projeto, documentando o mesmo, assim como uma análise estatística da sua execução. O mesmo está divido nas seguintes secções:

- **Arquitetura**: Identificação dos blocos funcionais e interfaces.
- **Estrutura do Código**: Descrição das APIs, principais estruturas de dados, principais funções e a sua relação com a arquitetura.
- **Casos de Uso Principais**: Identificação dos casos de uso e descrição da corrente de chamadas de funções.
- **Protocolo de Ligação Lógica**: Identificação dos principais aspetos funcionais da ligação lógica e descrição da estratégia de implementação destes aspetos.
- **Protocolo de Aplicação**: Identificação dos principais aspetos funcionais da camada de aplicação e descrição da estratégia de implementação destes aspetos.
- **Validação**: Descrição dos testes efetuados com apresentação quantificada dos resultados.
- **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 protocolo de comunicação, implementado de acordo com o guião fornecido, assenta-se em duas camadas independentes: a camada de ligação (*link layer*) e a camada de aplicação (*application layer*).

A camada de ligação (definida nos ficheiros `link_layer.c` e `link_layer.h`) especializa-se no estabelecimento da conexão, envio e receção de tramas de informação, e encerramento da conexão. É esta camada que verifica a validade da informação e resolve a ocorrência de erros de comunicação. As funções necessárias para estas tarefas são fornecidas por esta camada.

A camada de aplicação (definida nos ficheiros `app.c` e `app.h`) utiliza o serviço da camada de ligação e é responsável por tratar os dados a ser enviados/recebidos, gerando pacotes de controlo e pacotes de dados numerados.

Estrutura do Código

A estrutura do código relaciona-se diretamente com a arquitetura definida. Existem dois executáveis resultantes, um responsável pela transmissão (*writer*) e outro pela receção (*reader*) de dados. Estes utilizam as camadas definidas para cumprir a sua função.

Camada de ligação

- `llopen` - Configura a porta série e inicia a conexão de acordo com a *flag* que indica se está a ser utilizada pelo *writer* ou pelo *reader*.
- `llwrite` - É utilizada pelo *writer* para escrever tramas de informação.
- `llread` - É utilizada pelo *reader* para ler tramas de informação.
- `llclose` - Repõe as configurações da porta série e termina a conexão de acordo com a *flag* que indica se está a ser utilizada pelo *writer* ou pelo *reader*.
- `timeout_write` - Função auxiliar, utilizada para escrever uma trama e retransmitir a mesma um determinado número de vezes se não obter uma resposta dentro de um determinado intervalo de tempo.
- `nc_read` - Função auxiliar, utilizada para ler uma trama e responder adequadamente.

São ainda utilizadas outras funções para a geração e validação de tramas, assim como operações de *byte stuffing/destuffing*.

Camada de aplicação

- `make_data_package` - Gera um pacote de dados com as configurações necessárias
- `read_data_package` - Lê e interpreta um pacote de dados
- `make_control_package` - Gera um pacote de controlo com as configurações necessárias
- `read_control_package` - Lê e interpreta um pacote de controlo

Casos de Uso Principais

Os programas desenvolvidos têm como principal caso de uso a transferência de ficheiros entre dois computadores, através de uma porta série, sendo um computador responsável por ser o transmissor e outro responsável por ser o recetor.

Compilação e execução

Para utilizar os programas desenvolvidos, é necessário compilar os mesmos (através do comando `make`) e executá-los com os respetivos argumentos. De seguida encontram-se as instruções de execução dos mesmos.

Considera-se `wnc` como o executável resultante da compilação de `writenoncanonical.c` e `rnc` o executável resultante da compilação de `readnoncanonical.c`.

Instruções de execução do `wnc`:

```
./wnc serial_port file_path [-v]
```

- **serial_port**: obrigatório, indica o nº da porta de série
- **file_path**: obrigatório, indica o caminho do ficheiro a enviar
- **-v**: opcional, ativa o modo verboso

Exemplos:

- `./wnc /dev/ttyS1 pinguim.gif`
- `./wnc /dev/ttyS0 images/picture.gif -v`

Instruções de execução do `rnc`:

```
./rnc serial_port [file_path] [-v]
```

- **port**: obrigatório, indica o nº da porta de série
- **file_path**: opcional, indica o nome do ficheiro de destino (não estando definido, o ficheiro é guardado com o seu nome original)

- **-v**: opcional, ativa o modo verboso

Exemplos:

- `./rnc /dev/ttyS1`
- `./rnc /dev/ttyS1 -v`
- `./rnc /dev/ttyS0 picture.gif -v`

Sequências de chamada de funções

A transmissão dos dados segue a seguinte sequência:

1. O recetor e transmissor são iniciados numa porta série, definindo no transmissor o ficheiro a ser enviado.
2. É configurada a porta série e iniciada a conexão entre os dois computadores, com recurso à função `llopen`.
3. Estando a conexão iniciada, o transmissor envia um pacote de controlo, utilizando a `make_control_package` e `llwrite`, o qual é recebido pelo recetor através da `llread` e `read_control_package`.
4. O transmissor divide o ficheiro em tamanhos preestabelecidos e envia os vários pacotes de dados, com recurso à `make_data_package` e `llwrite`, os quais são recebidos pelo recetor pela `llread` e `read_data_package`.
5. Terminando a transferência de dados, é enviado novamente um pacote de controlo, que indica o fim da transferência.
6. Finalmente, a conexão é terminada e as configurações iniciais da porta série são repostas, com recurso à função `llclose`. As estatísticas da transferência são imprimidas.

Protocolo de Ligação Lógica

O Protocolo de Ligação Lógica foi implementado nos ficheiros `link_layer.h` e `link_layer.c`. O objetivo deste protocolo é fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão, neste caso, um cabo de série. A transmissão é organizada em tramas que podem ser de dois tipos:

- **Tramas de Supervisão** para comandos e respostas, com a estrutura: |F|A|C|BCC1|F|
- **Tramas de Informação** que servem de *wrap* para os dados a serem transportados, com a estrutura: |F|A|C|BCC1|Campo de Dados (vários octetos)|BCC2|F|
  - F = Flag (octeto `01111110`)
  - A = Campo de Endereço que pode assumir os valores:
    - `0x03` para comandos enviados pelo Emissor e respostas enviadas pelo Recetor
    - `0x01` para comandos enviados pelo Recetor e respostas enviadas pelo Emissor
  - C = Campo de Controlo um octeto que identifica o tipo de comando ou resposta
  - BCC = Block Check Character

Por sua vez as tramas de Supervisão podem conter diferentes comandos e respostas:

- **SET** - Comando enviado pelo Emissor para estabelecer conexão com o Recetor. Se o Recetor comunicar de volta, dá se início à transferência de dados. Tem no Campo de Controlo o octeto `0x03`
- **UA** - Resposta não numerada que pode ser enviada por qualquer interveniente. Tem no Campo de Controlo o octeto `0x07`
- **RR** - Resposta positiva enviada pelo Recetor caso tenha recebido uma trama de Informação válida. Tem no Campo de Controlo o octeto `R0000101`, em que R pode ser 1 ou 0 de acordo com o número de sequência da trama de Informação recebida.

- **REJ** - Resposta negativa enviada pelo Recetor caso tenha recebido uma trama de Informação inválida. Tem no Campo de Controlo o octeto `R0000001`, em que R pode ser 1 ou 0 de acordo com o número de sequência da trama de Informação recebida.
- **DISC** - Comando de desconexão que deve ser enviado inicialmente pelo Emissor e posteriormente pelo Recetor de modo a sinalizar o fim da comunicação e a ser iniciado o processo de fecho da porta de série. Tem no Campo de Controlo o octeto `0x0B`

O protocolo controla os erros através de várias medidas como o campo BCC, isto é Block Check Character, um octeto tal que exista um número par de 1s em cada posição/bit. É resultado de `xor`'s sucessivos aplicados a cada byte protegido pelo BCC em questão. Outras medidas incluem os pedidos de retransmissão e a identificação de tramas de Informação repetidas.

A transparência da transmissão é assegurada pela técnica de *byte stuffing*, que consiste em substituir os códigos, como por exemplo a Flag, por dois bytes: o de escape e um `xor` entre o código e o octeto `0x20`. Vale notar que o byte de escape também deve ser codificado caso ocorra algures numa trama. Esta estratégia permite que os diversos códigos possam ocorrer nas tramas sem prejuízo na interpretação das mesmas, já que podemos inverter o processo e fazer *byte destuffing* para reaver o estado original de uma determinada trama.

O protocolo assenta em 4 funções:

- `llopen` - Abrir a porta série
- `llwrite` - Escrever para a porta
- `llread` - Ler informação recebida na porta
- `llclose` - Fechar a porta série

A acrescentar a estas funções, existem outras auxiliares que são necessárias para a implementação deste protocolo, tais como:

- `timeout_write` - Escreve uma determinada trama para a porta que lhe é fornecida e retorna a resposta que recebe ou `NULL` se não recebeu uma resposta válida em nenhuma das tentativas que fez.
- `nc_read` - Lê uma trama na porta fornecida e manda a resposta adequada ao conteúdo que recebeu.
- `make_bcc` - Calcula o BCC para uma dada sequência de bytes, através da aplicação sucessiva de `xor`'s ao conjunto de bytes protegidos pelo referido BCC
- `make_info` - Cria uma trama de informação, a partir dos dados que lhe são fornecidos
- `byte_stuffing_count` - Retorna o número de instâncias em que o *byte stuffing* é necessário na trama que lhe é fornecida (conta o número de códigos)
- `byte_destuffing_count` - Retorna o número de instâncias em que o *byte destuffing* é necessário na trama que lhe é fornecida (conta o número de bytes de escape)
- `byte_stuffing` - Aplica o processo de *byte stuffing* na trama que lhe é fornecida
- `byte_destuffing` - Aplica o processo de *byte destuffing* na trama que lhe é fornecida

A função `timeout_write` é usada sempre que o Emissor envia um comando ou trama de Informação e necessita de esperar por uma resposta para continuar. Prentendemos que, após realizar o `write`, a função aguarde pela resposta e caso esta não seja recebida num determinado intervalo de tempo, a trama seja reenviada. Deste modo impedimos que ocorra um ciclo infinito e o protocolo torna-se mais robusto e resistente a falhas pontuais de comunicação. A função deve ainda realizar um número predefinido de tentativas (`tries`) destes reenvios antes de retornar `NULL`, resultando na interrupção do protocolo. Estes valores podem ser definidos nas macros `TIMEOUT` e `TRIES`, respetivamente. Para fazer este *loop*, utilizamos um `alarm`, como pode ser visto nos seguintes estratos de código:

```
// Como funcao global
void set_alarm()
{
  alarm_set = TRUE;
}

// No inicio da funcao timeout_write
```

```
(void)signal(SIGALRM, set_alarm);
write(fd, to_write, write_size);
alarm(TIMEOUT);

// No  final do loop que realiza os reads sucessivos e constroi a trama de resposta
    nfiefnwfeiofjefie fe jfioofjewfijeiiejo fejf fef ewf e few
if (alarm_set)
{
  alarm_set = FALSE;
  tries--;
  if (tries > 0)
  {
    write(fd, to_write, write_size);
    printf("Alarm triggered, trying again. %d tries left\n", tries);
    alarm(TIMEOUT);
  }
}
```

Por outro lado, a função `nc_read` é usada por parte do Recetor para ler e processar as tramas recebidas e enviar uma resposta adequada ao conteúdo ou comando recebido. Esta função inclui um ciclo onde realiza os `read`s e do qual só sai qundo lê uma trama válida, muito semelhante ao encontrado na `timeout_write`, exceto na ausência de *timeout* com `alarm`. Neste ciclo está implementado uma simples máquina de estados: inicialmente é ignorada qualquer informação até ser recebida a primeira Flag; a máquina permanece neste segundo estado até receber um byte que não seja uma Flag; por último são guardados todos os bytes que não sejam Flags até ser recebida novamente uma Flag. Se a informação recebida de facto se tratar de uma trama válida, a trama é processada e o seu tipo é identificado através de um `switch` com base no seu Campo de Controlo. No caso de ser um comando como SET ou DISC, a função limita-se a enviar a resposta adequada. Se, por outro lado, se tratar de uma trama de Informação, o segundo BCC é verificado, tal como já tinha sido verificado o primeiro e é feito o processo de *byte destuffing*, antes de enviar uma resposta. Em qualquer um dos casos, no final a trama lida é passada por referência para o argumento `read_package`.

A função `llopen` trata de configurar e abrir a porta de série. Recebe o identificador da porta e uma *flag* a indicar se se trata do Emissor (`TRANSMITTER`) ou do Recetor (`RECEIVER`). Faz uma configuração semelhante para os dois intervenientes, incluindo o `open` à porta. De seguida o Emissor envia o comando SET, ao qual o Recetor deverá responder com uma UA. Em caso de sucesso a função retorna o *file descriptor* da porta, caso contrário um número negativo que identifique o que correu mal.

A função `llread` lê uma trama de Informação e retira por referência o Campo de Dados da mesma para o argumento `buffer`. É resumidamente uma *wrapper* da função `nc_read` que já lê e processa a trama recebida pela porta de série e responde apropriadamente. A `llread` retorna um número negativo que identifica o erro, se o resultado da `nc_read` não for o esperado. Em caso de sucesso, retorna o tamanho do `buffer` após ser preenchido com o Campo de Dados da trama de Informação recebida.

A função `llclose` trata de fechar a porta de série. Recebe o *file descriptor* da porta e uma *flag* a indicar se se trata do Emissor (`TRANSMITTER`) ou do Recetor (`RECEIVER`). Para começar o Emissor deve mandar o comando DISC, ao qual o Recetor deve responder com um comando DISC também. O Emissor deve, depois, mandar um UA. Finalmente é chamada a função `close` para fechar a porta de série, após *resetar* as definições da porta para as anteriores à execução da `llopen`. Se alguma parte deste processo falhar, a função `llclose` retorna um número negativo identificativo do passo que falhou. Se tudo correr conforme o esperado 0 é o valor retornado.

Resumidamente, o Protocolo de Ligação Lógica permite estabelecer uma conexão entre duas máquinas, através de uma porta de série, e transferir informação entre elas, de uma forma eficiente, transparente e segura. Tudo isto com mecanismos de mitigação de erros e falhas de conexão. Em conjunto com uma camada de Aplicação, poderemos usar este protocolo para enviar um ficheiro entre duas máquinas.

Protocolo de Aplicação

O Protocolo de Aplicação foi definido nos ficheiros `app.h` e `app.c`. A aplicação suporta dois tipos de pacotes:

- **Pacotes de controlo** para sinalizar o início e o fim da transferência do ficheiro
- **Pacotes de dados** contendo fragmentos do ficheiro a transmitir

O ficheiro a ser enviado será dividido em pacotes de dados e estes pacotes serão divididos em tramas de Informação para serem enviados de acordo com o Protocolo de Ligação Lógica. Assim, o Protocolo de Aplicação tem apenas a responsabilidade de criar e interpretar estes dois tipos de pacotes, "ignorando" a estrutura das tramas referentes ao Protocolo de Ligação Lógica. Para tal contém quatro funções:

- `make_control_package` - Cria pacotes de controlo
- `read_control_package` - Processa pacotes de controlo
- `make_data_package` - Cria pacotes de dados
- `read_data_package` - Processa pacotes de dados

A função `make_control_package` cria os pacotes de controlo. Recebe um booleano, `start`, que indica se se trata de um pacote de sinalização de início (com '2' no campo de controlo) ou de final (com '3' no campo de controlo). Além disso, recebe o tamanho do ficheiro a ser transmitido e o nome do mesmo (nos argumentos `file_size` e `file_name` respetivamente), parâmetros que vão ser codificados com o formato TLV no pacote de controlo. O campo Type será 0 para o tamanho do ficheiro e 1 para o nome, por convenção. O pacote de controlo resultante será passado por referência para o argumento `control_package`. Esta função retorna o tamanho do pacote de controlo criado.

Por sua vez, a função `read_control_package` lê pacotes de controlo e interpreta as informações que estes fornecem. O argumento `control_package` representa o pacote de controlo a ser lido e `package_size` o seu tamanho em bytes. É verificado se o campo de controlo do pacote é válido, tal como se os campos Type são os esperados. Caso o pacote a ser lido seja de facto validado, os seus campos TLV são lidos, através de `memcpy`'s com os tamanhos indicados em cada campo Length. O tamanho do ficheiro indicado pelo pacote é guardado por referência no inteiro `file_size` e o nome no argumento `file_name`. É retornado o tamanho do pacote de controlo lido.

A função responsável por criar os pacotes de dados é a `make_data_package`. Numerar os pacotes de dados é outra responsabilidade desta função, como tal, após colocar o campo de controlo do novo pacote a 1, define o número de sequência como o módulo de 256 do contador que lhe é passado na variável `seq_n`. O tamanho do pacote é escrito em dois bytes, seguido da informação propriamente dita, ou seja, do fragmento do ficheiro a ser transmitido naquele pacote (passado no argumento `data`). O pacote de dados que resulta deste processo é passado por referência para o argumento `data_package`. O valor retornado corresponde ao tamanho o pacote de dados criado.

Por último, a função `read_data_package` lê e interpreta um pacote de dados. Inicialmente é verificado o campo de controlo do pacote fornecido como argumento, o `data_package`. Se o pacote passar nesta verificação, o número de sequência do pacote é guardado no argumento `seq_n`. Finalmente o campo de dados, cujo tamanho é calculado na expressão `data_package[2] * 256 + data_package[3]`, é lido e passado por referência para o argumento `data`. A função retorna o tamanho do campo de dados do pacote que leu.

A combinação destas 4 funções permite a criação e o processamento dos pacotes do nível da aplicação. A utilização deste protocolo em conjunto com o de ligação lógica (que contém as funções relacionadas com o envio de tramas) permite a transmissão e a receção dos dados do ficheiro.

Validação

Por forma a estudar a robustez dos programas desenvolvidos, foram realizados em laboratório testes de transmissão de ficheiros com as seguintes variações:

- Diferentes ficheiros, com diferentes tamanhos.
- Diferentes *baudrates*.

- Diferentes tamanhos máximos dos pacotes de dados.
- Interrupção temporária da ligação da porta série durante o envio.
- Introdução de ruído na porta série durante o envio.

Em todos os testes, os programas correram como esperado e a transmissão dos ficheiros foi bem sucedida.

Eficiência do protocolo de ligação de dados

Conclusões

O desenvolvimento deste trabalho laboratorial foi bem sucedido, tendo permitido a melhor compreensão dos assuntos estudados e da sua complexidade, através da implementação prática de um protocolo de dados. Foi possível compreender a importância da independência entre camadas, neste caso a camada de ligação e a camada de aplicação, permitindo transmitir informação entre dois computadores de uma forma segura e controlada, por meio de um cabo série.

Concluindo, todos os objetivos propostos foram cumpridos com sucesso, tendo estes contribuido para o nosso conhecimento e capacidades, no contexto da Unidade Curricular de Redes de Computadores.

Anexo I - Código fonte