1º Tabalho Laboratorial

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Turma 3 - Grupo C
Bernardo Barbosa - up201503477
Diogo Mota Pinto - up201404527
João Sá - up201506252

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

5 de Janeiro de 2018

Sumário

Este relatório tem como objectivo descrever o primeiro trabalho laboratorial da U.C. Redes de Computadores. Este foi desenvolvido usando a linguagem C, em ambiente Linux, sendo que tinha como fim criar um programa que permitisse transferir um ficheiro entre duas maquinas ligadas por uma porta de série assíncrona.

Foram cumpridos os objetivos deste trabalho, foi possível transferir dados entre dois computadores sem erros ou perdas, sendo possível restablecer a ligação após qualquer falha.

Conteúdo

1 Introdução **3**

2 Arquitetura **3**
2.1 Interface *(Command Line Interface)* 3
2.2 Camada de Ligação de Dados *(Data Link Layer)* 4
2.3 Camada de Aplicação *(Application Layer)* 4

3 Estrutura do código **4**
3.1 Principais funções 4
3.2 Estruturas de Dados 5

4 Casos de Uso Principais **6**

5 Protocolo de Ligação Lógica **6**
5.1 Principais aspetos funcionais 6
5.2 llopen() & llclose() 6
5.3 llwrite() & llread() 7

6 Protocolo de Aplicação **7**
6.1 Principais aspetos funcionais 7
6.2 connection() 8
6.3 send_file() & receive_file() 8

7 Validação **8**

8 Eficiência do protocolo de ligação de dados **9**

9 Conclusões **9**

1 Introdução

O trabalho teve como objetivo implementar um protocolo de ligação de dados, como explicado no guião, e o seu teste com a aplicação desenvolvida, via transferência de ficheiros através da porta de série. O relatório tem como objectivo expor os aspetos mais teóricos da realização do trabalho, que, dada a sua natureza teórica não terão sido avaliados na demonstração do projeto.

O relatório sera organizado da seguinte forma:

1. Arquitétura - Apresentação blocos funcionais e interfaces.
2. Estrutura do código - Apresentação das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura.
3. Casos de uso principais - Identificação dos casos de uso pricipais e sequência de chamada de funções.
4. Protocolo de ligação lógica - Identificação dos principais aspectos funcionais; descrição da estratégia de implementação destes aspectos com apresentação de extratos de código.
5. Protocolo de aplicação - Identificação dos principais aspectos funcionais e descrição da estratégia de implementação destes aspectos com apresentação de extractos de código.
6. Validação - Descrição dos testes efectuados com apresentação quantificada dos resultados.
7. Eficiência do protocolo de ligação de dados - Caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.
8. Conclusões - síntese da informação apresentada nas secções anteriores e reflexão sobre os objectivos de aprendizagem alcançados.

2 Arquitetura

O programa foi organizado em duas camadas que comunicam entre si e uma interface para o utilizador interegir com o programa.

2.1 Interface *(Command Line Interface)*

Todo o código relativo à interação com o utilizador, inicialmente recolhe os parâmetros de configuração, Recetor/Transmissor, número de *timeouts* permitidos, tempo necessário para um *timeout*.

Durante a execução da aplicação durante a transmissão de dados apresenta informação relativas ao progresso total *(bytes já transmitidos/ bytes totais)*. São também apresentadas mensagens de erro no caso do envio de uma trama falhar e de restabelecimento da ligação caso esta falhe por qualquer motivo.

2.2 Camada de Ligação de Dados *(Data Link Layer)*

A camada de Ligação de Dados, descrita nos ficheiros data_layer.c e data_layer.h, é a camada responsável pela comunicação das duas máquinas, funciona como ligação entre a camada de aplicação e a porta de série.

Contém funções que estabelecem e terminam a ligação, que escrevem e lêem os dados da porta de série, enquanto garantem o sincronismo e a numeração de tramas, e que detetam erros no *stuffing/destuffing* destas ou caso sejam danificadas durante a transferência por fatores externos no *BCC2*.

2.3 Camada de Aplicação *(Application Layer)*

A camada de Aplicação, descrita nos ficheiros app_layer.c e app_layer.h,é a camada responsável pela escrita e leitura do ficheiro, via emissão e receção de tramas, situada diretamente acima da camada de Ligação de Dados funciona como ligação entre esta e a interface com o utilizador.

3 Estrutura do código

3.1 Principais funções

Na camada de Aplicação *(Application Layer)* o protótipo das principais funções é:

```
int connection(const char *port, int mode);
int send_file(char* filename);
int receive_file();
```

A função connection() é responsável por validar a string que identifica o dispositivo de porta de série e por chamar a função llopen() que vai devolver o descritor do ficheiro da porta de série caso tenha sucesso.

A função send_file() é responsável por dividir o ficheiro em pacotes de dados que possam ser passados à camada de Ligação de Dados para serem processados e enviados, o ínicio da transferência é indicado com o envio de um pacote de controlo que contém as permissões, o nome e o tamanho do ficheiro a ser enviado.

A função receive_file() é responsável por receber pacotes de dados e juntar estes mesmos de maneira semelhante ao ficheiro original, o fim da transferencia é indicado com a receção de um pacote de controlo.

Na Camada de Ligação de Dados *(Data Link Layer)* o protótipo das principais funções é:

```
int llopen(int port, int mode);
int llwrite(int fd, unsigned char* data_packet, unsigned int
    data_packet_length);
int llread(int fd, unsigned char *data_packet, unsigned int
    *data_packet_length);
int llclose(int fd);
```

A função llopen() estabelece a ligação entre as duas máquinas (emissor e recetor) garantindo que está tudo pronto para se iniciar a trânsferencia de dados.

A função llwrite() é chamada pelo emissor, que envia uma trama de informação contendo os dados que lhe foram passados pela camada de Aplicação após algum processamento, e espera por uma confirmação de receção com sucesso.

A função llread() é chamada pelo recetor, e fica à espera de receber uma trama de informação, e depois de confirmar a sua validade responde de maneira adequada.

A função llclose() termina a ligação e fecha os recursos utilizados pela aplicação.

3.2 Estruturas de Dados

Na camada de Aplicação *(Application Layer)* foi usada a seguinte estrutura:

```
typedef struct applicationLayer{
      int fileDescriptor; //Serial port file descriptor
      int mode; //TRANSMITTER 0 / RECEIVER 1
} applicationLayer;
```

Na Camada de Ligação de Dados *(Data Link Layer)* foi usada a seguinte estrura:

```
typedef struct linkLayer {
  int fd; //Serial port file descriptor
  char port[20]; //Serial port device
  int mode; //TRANSMITTER 0 / RECEIVER 1
  struct sigaction new_action; //timer "start"
  struct sigaction old_action; //timer "stopped"
  unsigned int timeout; //Time to timeout
  unsigned int numTransmissions; //Maximum number of retries
} linkLayer;
```

4 Casos de Uso Principais

A aplicação para ser executada de maneira correta, necessita de um pârametro, uma string que identifique o dispositivo de porta de série (normalmente "/dev/ttyS0"). Depois de esse pârametro ser validado:

- Cabe ao utilizador selecionar o modo de execução (Transmissor ou Receptor), o número máximo de *timeouts*, e o tempo para ocorrer um *timeout*.
- Estabelece uma ligação.
- O emissor envia parte dos dados do ficheiro, e espera por uma resposta para continuar a enviar o resto dos dados.
- O receptor recebe os dados, valida os mesmos, responde ao emissor, de maneira a que este continue a enviar os dados e escreve no ficheiro de output.
- Termina a ligação.
- Fecha os recursos utilizados pela aplicação.

5 Protocolo de Ligação Lógica

5.1 Principais aspetos funcionais

- Configuração da porta de série.
- Estabelecer e fechar a ligação entre duas máquinas através da porta de série.
- Transferência de dados através da porta de série.
- Deteção e tratamento de erros durante a transferência de dados.

5.2 llopen() & llclose()

A função *llopen()* começa por configurar a porta de série com as características pretendidas.

Ao ser invocada pelo Emissor cria uma trama **SET** *(Setup)* e envia-a, ficando à espera de uma resposta - neste caso a trama **UA** *(Unnumbered Acknowledgment)* - proveniente do Recetor, para este fim as tramas são criadas na função *create_US_frame()*, enviadas usando a função *write_frame()* e recebidas usando a função *read_frame()*.
Depois da trama ser enviada é sempre activado um alarme que tem a duração selecionada pelo utilizador, caso durante o tempo definido a frame não for recebida ocorre um timeout, e reenvia a trama. Os timeouts são acumulativos, sempre que há um timeout o numero de timeouts é atualizado e caso passe o número seleccionado pelo utilizador a função retorna -1, de maneira a sinalizar que ocorreu um erro.

Ao ser invocada pelo Recetor, fica à espera até receber uma trama **SET** e quando isto acontece responde com uma trama **UA**, estabelecendo assim

a ligação com sucesso.

A função *llclose()* é utilizada para terminar a ligação previamente estabelecida.

Ao ser invocada pelo Emissor cria uma trama **DISC** *(Disconnect)* e envia-a, ficando à espera de uma resposta - neste caso a trama **DISC** - posteriormente cria e envia uma trama **UA**.
Após isto, repões as configurações anteriores da porta e termina com sucesso.

Ao ser invocada pelo Recetor, espera até receber uma trama **DISC**, respondendo com uma trama do mesmo tipo, posteriormente repõe as configurações anteriores da porta e termina com sucesso.
À semelhança de funções anteriores, também esta função recorre ao uso de um alarme para gerir a ocorrência de timeouts.

5.3 llwrite() & llread()

Estas são as funções responsáveis pela escrita e leitura de dados na porta de série, durante o decorrer do programa.

A função *llwrite()* é sempre invocada pelo Emissor, que lhe passa um pacote de dados e o seu tamanho, em seguida este pacote é encapsulado numa trama **I** *(Information)*, para este efeito é chamada a função *create_I_frame()* que trata de gerar todos os campos e de fazer o byte stuffing necessário. Em seguida envia esta trama.

A função *llread()* invocada pelo Recetor, espera até receber uma trama, caso esta trama seja **DISC** procede ao fecho da ligação estabelecida; caso não seja uma trama deste tipo, verifica a validade do cabeçalho da trama usando a função *is_I_frame_header_valid()*, caso seja inválido cria e envia a trama **REJ**, caso contrário faz o destuffing graças à função *read_byte_destuffing()*, verifica este valor com o *Block Check Caracter* correspondente ao pacote de dados, confirmando então se o pacote é valido. Caso o pacote seja de facto válido envia uma trama **RR** - mesmo o pacote sendo duplicado - caso contrário transmite uma trama **REJ**.

6 Protocolo de Aplicação

6.1 Principais aspetos funcionais

- Iniciar a conexão entre as duas máquinas.
- Criar e enviar os pacotes de dados e de controlo.
- Enviar e receber o ficheiro.

6.2 connection()

Depois de o utilizador configurar a aplicação com os parâmetros desejados, a função *connection()* é chamada e começa por validar a string que identifica o dispositivo de porta de série, em seguida chama a função *llopen()* passando como argumento o modo (Emissor/Recetor), a função llopen() estabelece uma ligação entre o Emissor e o Recetor e devolve o descritor do ficheiro da porta de série.

6.3 send_file() & receive_file()

A função *send_file()* chamada pelo Emissor, é responsável por dividir o ficheiro em pacotes de dados que possam ser passados à camada de Ligação de Dados para serem processados e enviados, o ínicio da transferência é indicado com o envio de um pacote de controlo (**Start Packet**) que contém as permissões, o nome e o tamanho do ficheiro a ser enviado. Para escrita é usada a função *llwrite()* da camada de Ligação de Dados.

A função *receive_file()* chamada pelo Recetor, é responsável por receber pacotes de dados e juntar estes mesmos de maneira semelhante ao ficheiro original, o fim da transferencia é indicado com a receção de um pacote de controlo (**End Packet**), para a leitura é usada a função *llread ()* da camada de Ligação de Dados.

7 Validação

Para testar o funcionamento da aplicação foram introduzidos alguns casos para confirmar que a detecção de erro e a resolução destes estava a ser correctamente implementada:

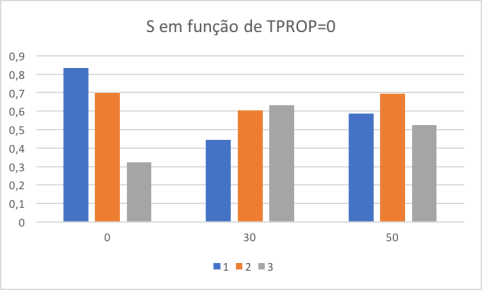
- Enviar um ficheiro;
- Enviar um ficheiro, interrompendo a transmissão carregando no botão de interrupção duas vezes;
- Enviar um ficheiro introduzindo erros na transferencia através de um cabo de cobre;
- Enviar um ficheiro, interrompendo a ligação, e introduzindo erros na transferência através do cabo de cobre, posteriormente resumindo a ligação.

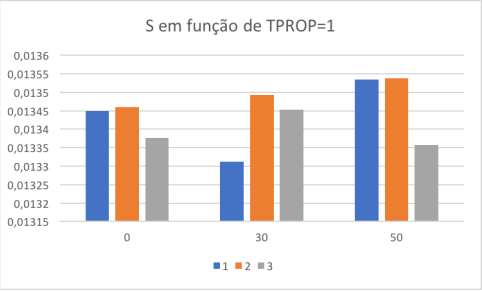
O programa foi capaz de detetar os erros e corrigi-los reenviando novamento os pacotes comprometidos, e de restabelecer a ligação quando esta foi perdida.

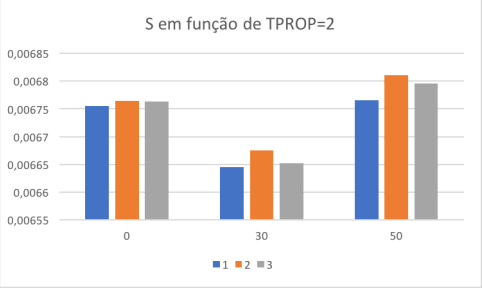
8 Eficiência do protocolo de ligação de dados

Para obter a eficiência do protocolo de ligação de dados, realizámos uma série de cálculos e experiências de forma a obter resultados precisos e realistas.

Sendo a capacidade da ligação correspondente a 38400 bit/s e partindo das equações R = Tamanho / tempo (com diferentes valores de TPROP e FER) e S = R/C, apresentamos 3 gráficos, correspondentes ao teste para TPROP igual a 0,1 e 2 para FER igual a 0,30 e 50.







Em cada gráfico, efetuamos 3 tentativas para cada configuração, de modo a obter uma melhor noção da eficiência do protocolo.

9 Conclusões

O objectivo de trabalho foi alcançado, pois foi possível de implementar um programa dividido em camadas independentes entre si com um protocolo de ligação de dados capaz de enviar um ficheiro de uma máquina para a outra de acordo com as especificações pretendidas, preparado para detetar erros e corrigi-los eficientemente.