

**Trabalho Laboratorial nº1:**

**Ligação de Dados**

**MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO**

Turma 3 Grupo x:

Antero Gandra, up201607926

Francisco Moreira, up201607929

Sumário

Realizado no contexto da unidade curricular de Redes de Computadores, foi requisitado aos alunos a implementação de um protocolo de comunicação assíncrona para a transmissão de um ficheiro através de uma porta série RS-232.

Na realização deste projeto tirámos várias conclusões sendo que as principais foram que (1) e que (2)

Introdução

Objetivos do trabalho

Objetivos do relatório:

O relatório encontra-se dividido nas seguintes secções:

* **Arquitetura** – onde se descrevem os diferentes blocos funcionais e interfaces;
* **Estrutura do código** – onde se descrevem as APIs, as principais estruturas de dados, as principais funções e a relação das funções com a arquitetura);
* **Casos de uso principais** - onde se identificam os casos principais de uso e as sequências de chamada de funções para cada;
* **Protocolo de ligação lógica**- onde se identificam os principais aspetos funcionais e se descreve a sua estratégia de implementação;
* **Protocolo de aplicação** - onde se identificam os principais aspetos funcionais e se descreve a sua estratégia de implementação;
* **Validação** - onde se descrevem os testes efetuados;
* **Eficiência do protocolo de ligação de dados**- onde fazemos uma caraterização estatística da eficiência do protocolo, com recurso a medidas sobre o código desenvolvido.
* **Conclusões** - onde resumimos a informação apresentada nas secções anteriores e refletimos sobre os objetivos de aprendizagem alcançados

Arquitetura

**Camadas de protocolo**

O trabalho está organizado em duas camadas – *layers* – que permitem o correto funcionamento da aplicação: a camada do *protocolo de ligação de dados* e a camada de *aplicação*, que estão implementados em diferentes ficheiros *source* e *header*. Os ficheiros link.ce link.h representam a camada de ligação de dados enquanto os ficheiros application.c e application.h representam a camada de aplicação.

A camada do protocolo de ligação de dados contém funções genéricas que relacionadas com a porta série, tratando do estabelecimento da ligação e da transferência de dados e deteção de erros.

A camada de aplicação por outro lado contém funções mais especificas relacionadas, neste caso, com a transferência de um ficheiro pela porta serie.

**Interface e opções**

A interface é proporcionada pelos ficheiros da camada da aplicação que indica informação sobre o estado da transferência do ficheiro. No entanto, a camada de ligação de dados pode por vezes colocar informação na interface quando ocorrem erros.

As opções de ligação como *baud rate, timeout, timeout tries* e tamanho máximo da mensagem podem ser alteradas no ficheiro settings.txt. Adicionalmente existe uma opção neste ficheiro que indica a possibilidade de ser gerado um erro aleatório em cada mensagem transferida. Esta opção foi usada para testar e comparar a eficiência do protocolo, deve obviamente ser colocada a 0 para uso real.

Estrutura do código

O código encontra-se dividido em 3 ficheiros correspondentes às 2 camadas da arquitetura e um outro ficheiro alarme.c que trata de processar timeouts gerados pelo sinal SIGALRM do sistema.

Naturalmente cada um destes ficheiros possui um header file correspondente onde estão declaradas as funções e estruturas de dados usadas, assim como algumas macros.

**Ligação de dados (link.c e link.h):**

* **Funções**:
  + **llopen**: estabelece a ligação entre o ‘writer’ e o ‘reader’.
  + **llclose**: fecha a ligação e entre o ‘writer’ e o ‘reader’.
  + **llwrite**: envia uma mensagem e espera pela resposta.
  + **llread**: lê uma mensagem e verifica os erros.
  + **stuff**: realiza o stuffing das mensagens.
  + **destuff**: realiza o destuffing das mensagens.
  + **processBCC**: calcula o BCC.
  + **receiveMessage**: lê uma mensagem. (é chamada pelo llwrite).
  + **identifyMessageControl**: identifica se uma mensagem tem o tipo previsto.
  + **sendCommand**: prepara e envia um comando.
  + **sendMessage**: prepara e envia uma mensagem.
  + **printStats**: imprime na linha de comandos as estatisticas da ligação.
  + **statisticsSetup**: inicializa as estatisticas da ligação.
  + **connectionSettings**: lê as settings pretendidas para a ligação.
* **Estruturas de dados:**
  + **State**: Estados da máquina de estados
  + **Mode**: Modo da aplicação, ‘writer’ ou ‘reader’
  + **Control**: Tipo do comando de controlo
  + **MessageType**: Tipo de mensagem
  + **ErrorType**: Tipo de erro
  + **Data**: lista de caracteres e tamanho da informação, usado em Message
  + **Message**: Informação de uma mensagem
  + **Stats**: Estatisticas da ligação
  + **Settings**: Opções da ligação

**Aplicação (application.c e application.h):**

* **Funções**:
  + **main**: Inicio do programa e processamento dos parâmetros
  + **sendFile**: Envio de um ficheiro
  + **receiveFile**: Receção de um ficheiro
  + **receiveControl**: Receção de um pacote de controlo
  + **receiveData**: Receção de um pacote de dados
  + **sendControl**: Envio de um pacote de controlo
  + **sendData**: Envio de um pacote de dados
  + **fileSize**: Lê o tamanho de um ficheiro
  + **openFile**: Abre um ficheiro
* **Estruturas de dados:**
  + **AppControlType**: Tipo de pacote
  + **Parameter**: Tipo de parâmetro no pacote de controlo (Size ou Name)

**Alarme (alarm.c e alarm.h):**

* **Funções**:
  + **alarmHandler**: Handler do alarme.
  + **setAlarm**: Configura o alarme.
  + **stopAlarm**: Para o alarme.

**Protocolo de** liga**ção lógica**

A camada de ligação de dados ´e a camada de mais baixo n´nível e ´e a camada responsável  
pela interação direta com a Porta Série. Algumas das funcionalidades implementada por  
esta camada são: abertura e fecho da Porta Série; escrita de um tramas de informação e controlo; leitura de um tramas de informação e controlo; criação de tramas de controle; *byte*  
*stuffing* e *byte destuffing* de uma trama; *framing* e *deframing* de uma trama.  
A nível da API da camada de ligação de dados foram implementadas as quatro funções previstas: **llopen, llclose, llread** e **llwrite**.

* A função **llopen** abre a ligação pela porta série com as configurações escolhidas e estabelece a ligação enviando um comando SET do ‘writer’ e aguardando pela resposta UA do ‘reader’.
* A função **llclose** estabelece o fim da ligação pela porta série trocando comandos DISC entre o ‘writer’ e o ‘reader’ e por fim um comando UA do ‘writer’ para confirmar o fim da ligação. Por fim as definições iniciais da porta série são repostas.
* A função **llread** ´e responsável pela leitura de informação da Porta Série, sendo que irá aplicar *destufffing* e *deframing* `a trama recebida. Se ocorrer um erro no *destuffing* ou um erro na *frame* que não seja no *BCC2* a trama ´e descartada, ficando assim `a espera de uma  
  nova trama. Se houver um error, e for no *BCC2*, a trama ´e descartada, o recetor continua a espera de uma nova trama, mas envia uma trama de controle do tipo REJ. Se a trama for corretamente recebida, esta ´e retornada e o recetor envia uma trama de controle do tipo RR.
* A função **llwrite** ´e responsável pelo envio de informação através da Porta Série. Esta recebe a mensagem a enviar da camada superior e aplica-lhe *framing* e *stuffing*. De seguida, tenta escrever a trama, sendo que se não receber uma resposta do tipo RR durante um  
  intervalo de tempo previamente definido - default ´e 3 segundos - este reenvia a trama. O reenvio da trama ´e feito um número de vezes previamente definido - default ´e 3 tentativas.  
  Se ao fim desse número de tentativas não tiver obtido sucesso, retorna erro.

Protocolo de aplicação

A camada de aplicação é de alto nível e trata do envio e da receção do ficheiro fonte através do API fornecido pela camada da ligação de dados.

**As funcionalidades implementadas por esta camada são:**

* iniciar a ligação;
* **se for emissor**: ler o ficheiro fonte e dividi-lo em pacotes de dados;
* **se for recetor:** recompor o ficheiro fonte através de pacotes de dados;
* enviar e receber os pacotes;
* terminar a ligação.

Foi assim desenvolvido o API (application.c) que gere a interação dos pacotes de dados

**Em relação aos pacotes de dados:**

* A função makeDataPacket que cria o pacote de dados a partir da informação original do ficheiro fonte;
* **sendData** que envia o pacote de dados (utilizando llwrite);
* **receiveData** que recolhe a informação do ficheiro original e recebe o pacote de dados (utilizando llread).

**Em relação aos pacotes de controlo:**

* A função makeControlPacket que cria o pacote de controlo;
* **sendControl** que envia o pacote de controlo (utilizando llwrite);
* **receiveControl** que recebe o pacote (utilizando llread).

**Funções implementadas:**

• **sendFile (emissor)** que inicia a ligação, lê o ficheiro fonte e divide em pacotes de dados, envia-os e termina a ligação. Utiliza 3 funções do API da ligação de dados: llopen, llwrite e llclose e API de pacotes.

• **receiveFile (recetor)** que inicia a ligação, lê os pacotes de dados, reconstrói o ficheiro fonte e termina a ligação. Utiliza 3 funções do API da ligação de dados: llopen, llread e llclose e API de pacotes.

Validação

Para validar do programa desenvolvido, garantindo, desta forma, que funcionava corretamente com o respetivo protocolo, foram realizados vários testes durante o desenvolvimento e demonstração do programa.

Foram testados vários ficheiros, de diferentes tipos e tamanhos e enviados com diferentes baudrates e tamanhos de pacotes de informação.

Também foram realizados testes de interrupção da comunicação na Porta Série e de introdução de erros através do curto-circuito existente nas portas.

Todos estes testes foram realizados no momento de avaliação, na presença do professor.

Conclusões

O protocolo implementado pode ser divido em duas camadas: LinkLayer e ApplicationLayer. A LinkLayer como sendo a camada de mais baixo n´nível e fazendo iteração direta com a porta série e a ApplicationLayer como sendo a camada de mais alto n´nível e que faz uso da API da LinkLayer para enviar / receber o ficheiro.

O protocolo foi implementado com robustez a erros e interrupções, conseguindo em ambos casos retomar a transferência com sucesso. A eficiência do protocolo ´e de cerco de 77%, aproximando-se bastante dos valores teóricos esperados.

No final, o grupo conclui que a implementação de protocolos se trata de um desafio devido á deteção de erros e de recuperação. Compreendemos agora os conceitos teóricos que estão implícitos neste trabalho, sendo eles o uso de stuffing, destuffing, framing, deframing e Stop&Wait com sequence numbers.