

Trabalho prático 1

Redes e Computadores

Ligação de dados

Trabalho realizado por:

Nuno Oliveira [up201806525@fe.up.pt](mailto:up201806525@fe.up.pt)

Luís Pinto [up201806206@fe.up.pt](mailto:up201806206@fe.up.pt)

Sumário

Este trabalho foi realizado no âmbito da disciplina de Redes de Computadores com o objetivo de estudar a transferência de dados através duma aplicação com recurso a um protocolo de ligação de dados. Tanto a aplicação como o protocolo de ligação e o controlo de erros foram implementados por nós.

POR AQUI ALGO DAS CONCLUSÕES

Introdução

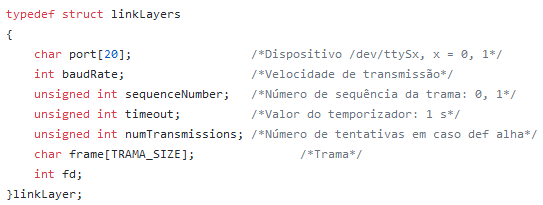
O trabalho prático pode ser dividido em duas grandes partes: desenvolver a aplicação de teste e desenvolver o protocolo de ligação de dados. O protocolo deve fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um cabo de série. Por sua vez, o objetivo da aplicação seria desenvolver um protocolo de aplicação relativamente simples para transferir ficheiros fazendo uso do protocolo de ligação de dados.

Este relatório visa explicar a nossa implementação das funcionalidades já mencionadas bem como a teoria do trabalho e é composto pelas seguintes secções (por ordem):

* Arquitetura - blocos funcionais e interfaces.
* Estrutura do código - APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura.
* Casos de uso principais - identificação; sequências de chamada de funções.
* Protocolo de ligação lógica - identificação dos principais aspetos funcionais; descrição da estratégia de implementação destes aspetos com apresentação de extratos de código
* Protocolo de aplicação - identificação dos principais aspetos funcionais; descrição da estratégia de implementação destes aspetos com apresentação de extratos de código.
* Validação - descrição dos testes efetuados com apresentação quantificada dos resultados, se possível.
* Eficiência do protocolo de ligação de dados - caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido. A caracterização teórica de um protocolo *Stop&Wait*, que deverá ser usada como termo de comparação, encontra-se descrita nos slides de Ligação Lógica das aulas teóricas.
* Conclusões - síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados.
* Anexos

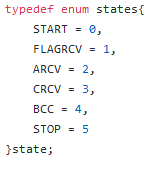
Arquitetura

Funcionalmente, tanto a aplicação como o protocolo têm duas maneiras distintas execução: como emissor do ficheiro e como recetor do mesmo. A interface dispõe de quatro funções, duas para estabelecer e terminar a ligação e outras duas para enviar e receber dados. Cabe à aplicação fazer uso dessas funções para estabelecer a conexão e transferir o ficheiro chamando as funções certas consoante o modo de execução.

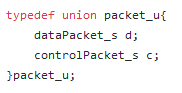
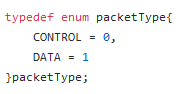
Estrutura do código

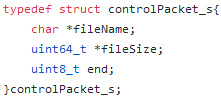
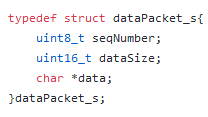
Principais estruturas de dados:

A struct linkLayers é usada pelo protocolo de ligação de dados, contendo vários valores importantes como a porta para onde escrever/ler, o *baudrate* para configurar a porta série, o número de sequencia para usar ao criar e ler tramas (também atualizado pelo protocolo), o tempo de espera do alarme, o número de tentativas de reenviar a mesma trama até gerar um erro e por fim o *file descriptor* da porta série depois de aberta.

Usamos também um enum que contém cada estado das várias máquinas de estados usadas neste projeto para simplificar a leitura das mesmas:

Na camada da aplicação usamos uma union para guardar cada pacote lido como de dados ou de controlo e um enum para facilitar a identificação desse mesmo pacote. Cada pacote será guardado na struct respetiva junto com a informação relevante para ser usada depois.





Ao longo do código fazemos também uso de múltiplas macros definidas no ficheiro macro.h.

A interface protocolo-aplicação dispõe das seguintes funções:

* int llopen() – estabelece a ligação entre o emissor e recetor.
* int llclose() – termina a ligação entre emissor e recetor.
* int llwrite() – escreve para a porta séria uma trama e fica à espera de resposta.
* int llread() – lê da porta série uma trama e envia uma resposta consoante o que leu.

As principais funções da aplicação para além da main que as chama são:

* int sendFile() – cria as tramas de controlo e informação relativas ao ficheiro, e passa-as a llwrite() uma a uma para serem enviadas.
* int receiveFile() – chama llread() para receber as tramas de controlo e informação uma a uma e tenta criar o ficheiro.
* int readPacket() – lê uma trama de controlo ou informação.
* void controlPacket() – cria um pacote de controlo.
* void dataPacket() – cria um pacote de dados.

O protocolo de ligação de dados para implementar as funções da interface protocolo-aplicação usa também:

* int byteStuff() – aplica o mecanismo de *byte stuffing* a uma trama.
* int byteDeStuff() – realiza a operação inversa de *byte stuffing*.
* int infoPacket() – cria uma trama de informação.
* uint8\_t getBCC2() – obtém o valor do BCC2 duma trama.
* int setTermIO() – configura a porta série.
* int setupLinkLayer() - preenche a struct linkLayer.
* int receive() – espera por uma trama de supervisão.
* int send\_receive() – envia uma trama de supervisão e aguarda resposta.
* void atende() – função que atende o sinal do alarme.
* void setHeader() – cria uma trama de supervisão ou não numerada.

Casos de uso principais

Os dois casos de uso são transmissão e receção de dados embora ambos sejam interdependentes.

Transmissão de dados:

* O utilizador corre o programa com os argumentos que achar necessários para além do obrigatório “1” em primeiro para correr em modo emissor.
* É chamada a função llopen() que estabelece a ligação entre emissor e recetor. Se não for possível o programa termina e retorna -1.
* Se llopen() tiver sucesso é chamada a função função sendFile() que cria as tramas de controlo e as tramas de informação lendo-as diretamente do ficheiro. Estas tramas são enviadas uma a uma usando llwrite() só enviando a próxima trama quando llwrite() retornar sucesso (um valor igual ou superior a 0).
* Se tudo funcionar corretamente no fim é chamada llclose() para terminar a ligação.

Para o caso de receção dos dados muda apenas o argumento obrigatório (passa a ser 0) e a função chamada após llopen() retornar sucesso. Essa função é a receiveFile() que por si chama llread() para receber os conteúdos escritos na porta série por llwrite() e enviar uma resposta positiva ou negativa dependendo do conteúdo da trama. Se as tramas chegarem com sucesso ao receiveFile() este tentará criar o ficheiro e no fim chamará llclose() identicamente à parte de transmissão de dados.

Exceções incluem falhas no estabelecimento da ligação que implicam terminar a aplicação e erros a tentar enviar/receber o ficheiro que são maioritariamente tratados pelo protocolo de ligação, mas se persistirem eventualmente lidariam a erros no ficheiro pelo que também terminam a aplicação e retornam um valor de erro.

Protocolo de ligação lógica

No protocolo de ligação de dados foram implementadas as quatro funções pedidas.

Llopen(): Esta função recebe como argumentos o modo em que o processo está correr bem como a porta de onde pretende ler/escrever. Dependendo do modo de execução serão executados blocos de código diferentes. Inicialmente não há diferenças nos blocos pois é necessário preencher a struct linkLayer, abrir efetivamente a porta série e configurá-la. Só depois os blocos diferem dado que o emissor envia uma trama de supervisão com o valor SET e fica à espera duma resposta com o valor UA usando a função send\_receive(). Já o recetor fica inicialmente à espera do SET usando a função receive() e se esta retornar com sucesso escreve então a trama com o valor de UA. Se algum destes passos não correr como esperado a função simplesmente retorna -1 indicando erro caso contrário a conexão foi estabelecida com sucesso.

Llwrite(): Recebendo a linkLayer em que deve trabalhar, um array que contem um pacote de dados ou controlo e o tamanho do mesmo, o llwrite() começa por subscrever o alarme e declarar as variáveis necessárias para tratar esse array como a “packet” para adicionar o bcc2 e a “stuffedPacket” para onde se vai escrever o array ao fazer byte stuffing e adicionar os cabeçalhos. Assim o primeiro passo é obter e escrever o BCC2. A seguir é feito então o byte stuffing e com o resultado é criado um pacote de informação adicionando os cabeçalhos pertinentes. É atualizado o número de sequência na linkLayer para continuar a usar corretamente na próxima escrita. Após isto tudo a trama está finalmente pronta a ser enviada e usando o alarme fica-se à espera duma resposta que deve conter o valor RR durante o tempo guardado na linkLayer. Caso não receba resposta após o sinal do alarme ou a resposta contenha REJ em vez de RR a função tenta escrever a trama novamente e este ciclo é repetido um determinado número de vezes também guardado na linkLayer até que receba a resposta positiva para poder retornar o número de carateres escritos indicando sucesso. Se isto não se verificar é retornado -1 indicando erro.

Llread(): A esta função é passado o linkLayer que deve ser usado bem como o apontador para o array onde deve ser escrita a trama lida após esta tenha sido tratada. A primeira coisa que se encontra é uma máquina de estados que lê e guarda a informação da trama até receber a segunda flag (0x7E) que aponta o fim da trama. Após sair da máquina de estados é feito o *destuffing* da informação e verificação de que recebe o mesmo BCC2 que obtém pela trama lida para garantir que não há erros de qualquer natureza. Se tudo correr bem é escrita uma trama de supervisão com o valor RR e retorna o número de carateres lidos para indicar sucesso. Se houver erro em algum destes passos em vez de RR a trama escrita terá o valor REJ e volta-se à máquina de estados para tentar ler a mesma trama na esperança que chegue corretamente desta vez.

Llclose(): Identicamente ao llopen() esta função recebe o modo de execução e consoante o valor executa um bloco diferente de código. Se for emissor envia uma trama de supervisão com o valor DISC e fica à espera de uma resposta com o mesmo valor para voltar a enviar uma trama desta vez como valor UA e terminar. Já o recetor, como foi dito, é suposto receber o DISC e enviar outro. Ambas as funções usam a função send\_receive() para ler e escrever. Finalmente, a configuração da porta é mudada para o seu estado inicial e é fechada a porta série.

As funções send\_receive() e receive(): estas funções são semelhantes no sentido em que ambas fazem uso duma máquina de estados para ler uma trama de supervisão e retornam 0 em caso de receber o valor esperado ( passado como segundo argumento). A diferença está no facto da primeira função primeiro escrever uma trama de supervisão com o valor recebido no terceiro argumento e ficar à espera duma resposta usando o alarme.

O uso de alarme nas funções send\_receive() e llwrite(): O alarme é usado da mesma maneira nas duas funções para se poder esperar um determinado tempo por uma resposta, guardado na linkLayer. Assim nas duas é subscrito o alarme tendo como handler do sinal a função atende(). Esta função quando executada coloca uma flag a 1 e itera a variável que contém o número de tentativas de leitura. Assim a função que está a usar o alarme apenas tenta ler da porta quando a flag estiver a 1 e se for preciso volta a iniciar o alarme e coloca a flag 0 para tentar ler novamente até que o número de tentativas iguale o que está guardado na linkLayer.

Protocolo de aplicação

sendFile(): Esta função recebe a linkLayer a passar ao llwrite() e o nome do ficheiro a enviar. Com esta informação o primeiro passo é abrir o ficheiro para que se possa lê-lo e criar o primeiro pacote de controlo com as informações relevantes para indicar o inicio do ficheiro passando-o para à função llwrite(). Após confirmado o envio do pacote de controlo está na altura de ler bloco a bloco do ficheiro, criar um pacote de dados com o bloco lido, enviar passando novamente para llwrite() e ficar à espera do retorno. O próximo bloco só é enviado se não ocorrerem erros na parte do protocolo de ligação. Cada bloco tem 256 bytes como default tirando o último que pode obviamente ter menos. Depois de enviado o ficheiro na integra é enviado outro pacote que simboliza o fim do ficheiro.

receiveFile(): O receiveFile() usa a função readPacket() para processar os pacotes que recebe e consiste apenas dum ciclo que chama continuamente llread() para tentar ler algo. Se o pacote for de controlo que marca o início de um ficheiro, é criado um ficheiro com o nome indicado no pacote, se for um pacote de dados, os dados são escritos no ficheiro e se for um que marca o fim do ficheiro o ciclo é terminado.

Validação

Testes efetuados:

* Envio do pinguim.gif presente no moodle com 10968B
* Envio de um png com 504KB
* Envio de um ficheiro de vídeo com 8.8MB
* Geração de ruído na ligação
* Interrupção da ligação
* Introdução de erros aleatórios nas tramas
* Variação dos valores do baudrate, tamanho da trama, probabilidade de geração de erros aleatórios

Resultados:

Com uma trama de tamanho 512B e velocidade de propagação de 115200 bits/s um ficheiro vazio, o gif, png e vídeo mencionados demoram respetivamente 0.015s, 1,021s, 46,262s, 810s. Estes valores são ligeiramente superiores ao valor teórico se dividir tamanho pela velocidade mas é preciso ter em conta que é feito byte stuffing e que o emissor às vezes tem de esperar o tempo definido no alarme pela resposta devido ao tempo de processamento da trama pela parte do recetor. O protocolo responde bem também à geração de ruído e interrupções na ligação devido ao controlo de erros e ao mecanismo Stop&Wait conseguindo obter o ficheiro intacto. Funciona também com qualquer tamanho da trama desde que seja superior a 0 e inteiro.

Eficiência do protocolo de ligação de dados

No calculo da eficiência por default usamos o tamanho da trama a 256B, um baudrate de 38400 bits/s, sem atrasos no processamento da trama e 10% de probabilidade de geração de erro na mesma exceto no campo que está a ser efetivamente avaliado por secção. Importante também mencionar que a eficiência será sempre inferior a 1 por vários motivos como, por exemplo, escrever mais bytes do que realmente existem devido a byte stuffing, aos tempos de processamento e de espera por respostas. Para diminuir os erros estatísticos cada valor usado na tabela é resultado da média de 5 medições.

Variação do FER:

A eficiência diminui quanto maior for a probabilidade de erro dado que a quantidade de tramas a reenviar aumenta proporcionalmente, em média. No entanto, existe a mesma probabilidade de cada trama reenviada ter erro o que justifica os valores extremamente baixos de eficiência.

Variação de T\_prop (atrasos de propagação simulados):

Obviamente se houver atrasos na propagação o número efetivo de bits lidos por segundo diminui dado que a percentagem de tempo em que o protocolo está a processar dados e enviar/receber os mesmos diminui.

Variação da capacidade de ligação:

Com a observação do gráfico pode-se concluir que a capacidade de ligação não afeta substancialmente a eficiência do protocolo de ligação de dados.

Variação do tamanho da trama:

Como seria de esperar a eficiência do protocolo aumenta com o aumento do tamanho da trama. A partir de 256 bytes nota-se um estagnar da eficiência porque os cabeçalhos passam a representar uma pequena percentagem do que é realmente escrito. Para exemplificar, quando a trama tiver tamanho 1, são escritos 11 bytes de cada vez, 10 dos quais são cabeçalhos. Nota-se também umas variações mínimas à medida que se aumenta a trama pois apesar do peso do cabeçalho diminuir sempre que houver um erro, maior será a quantidade de bytes a escrever. Assim, de maneira a maximizar a eficiência o tamanho da trama teria que ser um meio termo entre menor peso dos cabeçalhos possível e menor quantidade de bytes a escrever em caso de erro para cada situação.

Conclusões

Anexos

