

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.

O trabalho realizado permitiu alargar os conhecimentos sobre a gestão da comunicação entre computadores, os vários parâmetros que influenciam a velocidade e a fiabilidade da transmissão, a gestão dos erros e formas de manter a integridade da informação enviada.

Introdução

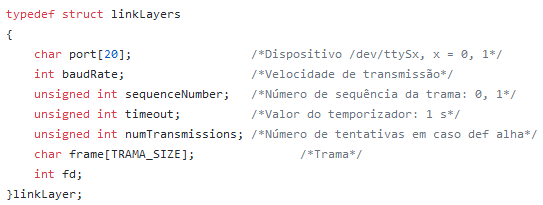
O trabalho prático pode ser dividido em duas grandes partes: desenvolver a aplicação de teste e desenvolver o protocolo de ligação de dados. O protocolo deve fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um cabo de série. Por sua vez, o objetivo da aplicação seria 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 e a caracterização teórica de um protocolo *Stop&Wait.*
* Conclusões - síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados.
* Anexos

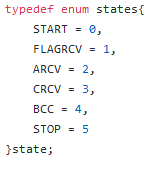
Arquitetura

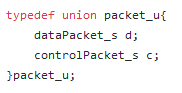
Funcionalmente, tanto a aplicação como o protocolo têm duas maneiras distintas execução: como emissor do ficheiro e como recetor do mesmo. A interface dispõe de quatro funções. Duas para estabelecer e terminar a ligação e outras duas para enviar e receber dados. Cabe à aplicação fazer uso dessas funções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

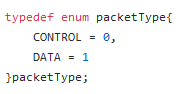
Estrutura do código

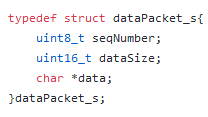
Principais estruturas de dados:

A struct linkLayers é usada pelo protocolo de ligação de dados. Esta contem vários valores importantes como o caminho do ficheiro da porta série, o *baudrate* da porta série, o número de sequencia para usar ao criar e ler tramas , o tempo de espera do alarme, o número de tentativas máximo de envio de uma trama o *file descriptor* da porta série depois de aberta. As estruturas são gravadas numa lista para que seja possível guardar as configurações das várias ligações de dados.

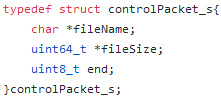
É também usado um enum que contém cada estado das várias máquinas de estados usadas neste projeto para simplificar a leitura das mesmas:







Na camada da aplicação é usada 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 faz-se 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, que são passadas à 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. Devolve no argumento packet a union que contém a informação lida
* 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 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 do tipo :   
  . /app <0 para sinalizar o emissor> <caminho do ficheiro a enviar> <numero da porta a usar> [tamanho da trama]

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 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 é enviada a próxima trama quando o llwrite() retornar sucesso (um valor igual ou superior a 0). No fim é chamada a llclose() para terminar a ligação.

Para o caso da receção de dados o argumento é do tipo:   
 ./app <1 para sinalizar o recetor> <numero da porta a usar> [tamanho da trama]

Depois de chamado o llopen() é chamada a função receiveFile() que chama o llread(), recebendo os conteúdos escritos na porta série. Se as tramas chegarem com sucesso é escrito no ficheiro de destino o seu conteúdo, No final é chamado o llclose() para fechar a conexão.

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.

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 devolvendo o ID da ligação

llwrite(): Recebendo o ID da ligação, 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. O primeiro passo é obter e escrever o BCC2. A seguir é feito o byte stuffing e com o resultado é criado um pacote de informação adicionando os cabeçalhos adequados. É atualizado o número de sequência na linkLayer. Após tudo isto 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, a função tenta escrever a trama novamente. Este ciclo é repetido no o número de vezes guardado na linkLayer out até que receba a resposta positiva. Em caso de sucesso é devolvido o número de caracteres escritos, caso contrário é devolvido -1, indicando erro.

llread(): A esta função é passado o Id da conexão bem como 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. A função retorna à máquina de estados de forma a ler de novo a trama que será recebida.

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 recebe o DISC e envia outro DISC, terminando de seguida. Ambas as funções usam a função send\_receive() para ler e escrever.

As funções send\_receive() e receive(): estas funções são semelhantes, ambas fazem uso duma máquina de estados para ler uma trama de supervisão e retornam 0 em caso recebam o valor esperado ( passado como segundo argumento). A diferença está no facto da primeira função escrever uma trama de supervisão com o valor recebido no terceiro argumento e só depois entra na máquina de estados.

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. A função que está a usar o alarme apenas lê da porta série, leitura esta feita num loop infinito. Caso a flag seja posta a 1 pelo alarme é escrita de novo na porta séria a informação sobre a qual se espera confirmação de receção.

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 o ficheiro bloco a bloco. É criado um pacote de dados com o bloco lido, enviado e este é enviado pela função llwrite llwrite() e ficar à espera do retorno. O próximo bloco só é enviado após o protocolo de ligação terminar com sucesso. Cada bloco tem blocos de tamanho definido no inicio do programa tirando o último que pode ter menos. Depois de enviado o ficheiro na integra é enviado outro pacote de controlo que simboliza o fim do ficheiro.

receiveFile(): O receiveFile() consiste num ciclo que chama continuamente llread() para ler bytes recebidos, usando ma máquina de estados. Depois de lidos os bytes de um pacote é usada a função readPacket() que processa os pacotes que recebe. 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.

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 o vídeo mencionados demoram respetivamente 0.015s, 1,021s, 46,262s, 810s. O valor teórico para o gif seria 0.76s, valor que corresponde a 75% do valor obtido mas é preciso ter em conta que para alem dos cabeçalhos é feito byte stuffing o que aumenta o valor real de bytes a enviar. 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 foi usado 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. 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 e cabeçalhos, 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. De notar que utilizando as equações associadas ao protocolo Stop&Wait, estes valores default definem o teto máximo de eficiência como 0.9, Assumindo o tempo de propagação 0 e probabilidade de erro 10%(0.1).

Estão em anexo os valores específicos associados aos gráficos a seguir apresentados.

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 os mesmos diminui uma vez que passa parte do tempo à espera de receber informação, sem a receber na realidade.

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, a quantidade de bytes a escrever aumenta. Assim, de maneira a maximizar a eficiência o tamanho da trama deverá ser um meio termo entre o menor peso dos cabeçalhos possível e a menor quantidade de bytes a escrever em caso de erro para cada situação(tamanho da trama).

O protocolo ARQ usado é o Stop and Wait. Do lado do transmissor, é enviada a trama e espera-se a resposta de confirmação. Do lado do recetor, analisa-se a trama recebida e envia-se uma resposta consoante a validade dos dados recebidos. Caso o transmissor não receba a confirmação num espaço de tempo estipulado, a trama é de novo enviada. Este protocolo é caracterizado pela sua simplicidade apesar de trazer várias limitações associadas à baixa eficiência.

Conclusões

O trabalho realizado permitiu alargar os conhecimentos sobre a gestão da comunicação entre computadores, os vários parâmetros que influenciam a velocidade e a fiabilidade da transmissão, a gestão dos erros e formas de manter a integridade da informação enviada.

Permitiu também conhecer a base para os protocolos de ligação e de aplicação usados para qualquer acesso a recursos na internet, tal como a relação entre a velocidade real e a influência de valores como os atrasos de propagação e a quantidade de erros ocorridos que juntos definem a verdadeira velocidade de qualquer ligação.

Foi também necessário melhorar os conhecimentos em C nos campos de gestão de memória, escrita em ficheiros e gestão de apontadores de forma a gerar código eficiente e que introduzisse o menor *overhead* possível.

Anexos

