Relatório

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Turma 3
Pedro Daniel dos Santos Reis
Vicente Fernandes Ramada Caldeira Espinha

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

25 de Novembro de 2017

Sumário

No âmbito do primeiro trabalho laboratorial da cadeira, foi pedida a elaboração de um programa capaz da transferência correta de dois ficheiros.

Iremos estabelecer conclusões sobre a eficiência dos nossos protocolos, juntamente com os problemas que advêm da metodologia utilizada.

Introdução

O objetivo final deste primeiro trabalho era a criação de um *software* capaz de passar através de uma porta série um ficheiro, detetando erros durante a execução.

O relatório está dividido em nove partes, cada uma descrevendo um aspeto do trabalho. A ordem é a seguinte:

- *Arquitetura* - Informação sobre os blocos funcionais e a interface do programa.
- *Estrutura do Código e Casos de Uso Principais* - Estruturas de dados utilizadas, principais funções, e identificação de sequências de chamadas, com excertos de código.
- *Protocolo de Ligação Lógica* - Identificação dos principais aspetos funcionais da camada e a sua estratégia de implementação, juntamente com código representativo.
- *Protocolo de Aplicação* - Semelhante à secção do protocolo de Ligação, agora na camada acima, a camada da Aplicação.
- *Validação* - Descrição dos testes efetuados para testar a fiabilidade do programa.
- *Eficiência do Protocolo de Ligação de Dados* - Apresentação estatística da eficiência do programa, comparativamente ao valor teórico de um protocolo Stop&Wait.

- *Conclusões* - Resumo do relatório, seguido de uma breve reflexão sobre os objetivos alcançados.

1 Arquitectura

A transmissão de informação está organizada em tramas, sendo estes *arrays* a estrutura base que é enviada. A trama tem um ou dois segmentos no caso das tramas de informação que apenas servem para confirmar que o conteúdo está correto, denominados BCC1 e BCC2.

O *software* está organizada em duas camadas, e dentro dessas duas temos ficheiros distintos para o emissor e o recetor.

Na primeira camada, a de aplicação, é realizada a leitura de dados, mais a abertura e fecho da ligação porta série e ficheiro a passar. Do lado do emissor ocorre a criação e dos pacotes de controlo, e dados extraídos do ficheiro. Estes são então passados para a camada de ligação do emissor, que após empacotar a informação numa trama a envia.

Com a informação agora transmitida, cabe ao recetor processá-la e enviar uma resposta ao emissor. Primeiramente analisa a trama, e caso não contenha erros, escreve para o ficheiro, enviando depois uma resposta RR, indicando ao emissor que tudo correu bem. Caso contrário, envia uma mensagem REJ, rejeitando a trama recebida e pedindo um reenvio.

Para iniciar o programa, o utilizador terá que executar dois comandos distintos para cada programa. No computador que tratará da emissão do ficheiro, terá de chamar o executável criado pelo *makefile*, utilizando como parâmetros a porta a ser utilizada, seguido do nome do ficheiro a enviar. O comando para o recetor é mais simples, apenas necessitando chamar o executável e escrever como único argumento a porta.

Durante a execução do programa, o utilizador poderá ver o progresso da transmissão, indicando o número de bytes que já foram enviados, e as respostas entre o emissor e recetor.

2 Estrutura do Código e Casos de Uso Principais

Foram utilizadas duas estruturas de C, a *stat*, usada para guardar informação sobre ficheiros, e a *termios*, que descreve uma interface geral para controlar portas de comunicação assíncronas. É também incluída a biblioteca *"time.h"*, para simular um tempo de propagação entre o envio e a receção das tramas.

Para cronometrar um tempo limite, é intercetada a chamada de sistema *SIGALRM* com um *handler*, necessitando assim das bibliotecas para esse efeito.

Seguindo a estrutura definida na arquitectura do programa, as principais funções na camada de ligação de dados chamadas da camada da aplicação são as que se seguem:

- *llopen()* - Estabelece-se a ligação entre as duas portas.
- *llwrite(int fd,unsigned char* buf,int size)* - Utilizada pelo emissor para enviar os dados que recebe como argumentos até receber receber mensagem de confirmação da sua validade.
- *llread(int fd)* - Utilizada pelo recetor com o intuito de ler uma trama, enviando uma determinada resposta caso tenha chegado com sucesso ou não.
- *llclose()* - Termina a ligação entre o emissor e o recetor, fechando a porta série e ficheiros abertos durante a execução do programa.

Uma outra função de relevo usada pelos dois processos em simultâneo é *int frread(int fd, unsigned char * buf, int maxlen)*, que vai lendo informação pela porta série até receber uma trama completa. Esta é depois processada numa função *int processframe(int fd, unsigned char* buf, int n)* que existe nos dois serviços, mas com implementações distintas, visto que existem diferenças específicas de recetor para emissor.

As principais funções implementadas na camada da aplicação, ambas do emissor, foram as seguintes:

- *buildControlPacket(char state,char* filename,int sizeFile)* - coloca num *array* a informação do nome e do tamanho do ficheiro. Esta é utilizada pelo recetor.

- *buildDataPacket(char* buf,int sizeFile)* - Cria um pacote, contendo dados extraídos diretamente do ficheiro, juntamente com o informação do número de sequência e número de octetos dos dados enviado.

3 Protocolo de Ligação Lógica

Neste protocolo é tratada de toda a transmissão e processamento entre os dois computadores, e há duas implementações desta camada, uma para o emissor e outra para o recetor.

No emissor, após a ligação de porta série ter sido estabelecida, é necessário iniciar a conexão entre os dois processos. Para isto o emissor emite uma comando SET pela porta. O recetor receberá os bytes, e inicia o processamento após receber a *flag* de fim de trama. Verificam-se os erros desta, e se não se encontrar nenhum, esta sequência de bytes entra na função mencionada na secção anterior, o *processFrame*. Segue-se um excerto dessa função do recetor.

```
if (buf[0] == FLAG && buf[1] == ADDRESS2 && buf[2] == SET
  && buf[3] == SET^ADDRESS2 && buf[4] == FLAG) { //Check if SET
    STOP = TRUE;//closes llopen
    return 0;
}
else if(buf[0] == FLAG && buf[1]== ADDRESS2 && buf[2] == DISC
          && buf[3] == DISC^ADDRESS2 && buf[4] == FLAG) { // Check if DISC
    STOP = TRUE; //closes llread
    return 0;
}
else if(buf[0] == FLAG && buf[1] == ADDRESS1 && buf[2] == UA
           && buf[3] == DISC^ADDRESS1 && buf[4] == FLAG) { // Check if UA
         STOP = TRUE; //closes llclose
         return 0;
}
```

Descrito acima está o que o programa faz quando lê determinadas mensagens do emissor. A variável STOP informa a camada de ligação do recetor que uma mensagem foi processada, e a resposta adequada pode ser enviada. No caso de receber um SET, o recetor enviaria um comando UA.

Está implementada do lado do emissor um temporizador para caso fique blo-

queado à espera de uma mensagem de resposta demasiado tempo. Por defeito este tempo é três segundos, mas pode ser alterado. Caso este tempo seja atingido, o emissor reenvia a mensagem e repete o processo de espera. Pode haver até três destes eventos, senão o emissor aborta a operação e termina a sua execução.

Após a ligação ter sido estabelecida, a camada de ligação de dados do emissor começa a receber num *buffer* a informação que tem de enviar. A função *llwrite()* é chamada, mas antes de transmitir uma trama faz um pré-processamento chamado *stuffing*, consistindo em substituir o octeto de saída *0x7e* por *0x7d 0x5e* e o octeto *0x7d* por *0x7d 0x5d*. Isto é necessário para impedir a existência de *flags* de saída a meio da trama. Se isto não for verificado, o recetor vai parar de ler quando encontrar a *flag*, mesmo que esteja a meio do campo de dados.

Após este processo, transmite a trama, acabada dentro da própria função. O recetor lê, e antes de verificar os erros, faz o *destuffing*, que é o processo inverso ao anteriormente descrito.

Este ciclo repete-se no emissor, até a camada acima acabar de passar toda a informação para esta, e invoca a função *llclose()*, também já descrita anteriormente. No recetor, ele fica permanentemente a ler da porta e a responder até ao momento em que é recebido um comando DISC, enviado no *llclose()* do emissor.

4 Protocolo de Aplicação

A camada de aplicação tem como função principal abrir os ficheiros, estabelecer a ligação na porta série e controlar todos os passos da execução do programa.

Semelhantemente ao caso anterior, há grandes distinções entre o lado do emissor e receptor. O último tem um ciclo de execução relativamente pequeno, tendo uma sequência descrita no código seguinte.

```
if(llopen(fd) < 0) {
      printf("ERROR in llopen! \n");
    else{
        if(llread(fd) < 0)
                printf("ERROR in llwrite! \n");
        else{
          if(llclose(fd) < 0)
                  printf("ERROR in llclose! \n");
```

```
            else
                    return 0;
            }
    }
}
```

O recetor fica preso num loop de leitura no *llread()* até a camada de ligação processar de um comando DISC do recetor, saindo assim da função e iniciando o processo de desconexão.

O emissor é também responsável pela transmissão para a camada de ligação dos dados do ficheiro. Tendo em conta que não se pode enviar o ficheiro todo numa trama, passamos toda os octetos do ficheiro para um *buffer*, e depois vamos guardando num segundo *buffer* o pacote que vamos mandar para o recetor. Guardamos numa variável global *transmittedData* o número de bytes que já foram transmitidos, a informação do tamanho do ficheiro numa *struct* st, e *MAX_SIZE* é o tamanho máximo de uma trama. O código é o que se segue.

```
unsigned char* buf = (unsigned char*) malloc (st.st_size);
fread(buf,sizeof(unsigned char),st.st_size,file);     //read from file
int sizebuf;
while(transmittedData < st.st_size){
        //calculates the size of the sizebuf
        if ((st.st_size - transmittedData) > MAX_SIZE)
                sizebuf = MAX_SIZE;
        else
                sizebuf = st.st_size - transmittedData;
        //allocates memory to the buffer
        unsigned char* buftmp = (unsigned char*)malloc (sizebuf);
        // copies part of the file to buftmp
        memcpy(buftmp,buf + transmittedData, sizebuf);
    //creates data packet
        unsigned char * CTRL_DATA =buildDataPacket(buftmp,sizebuf);
        free(buftmp);
        if(llwrite(fd,CTRL_DATA,sizebuf+4) < 0)  //send control data packet
                printf("ERROR in llwrite data! \n");
    //adds to the variable transmittedData what was sent
    if (transmittedData + sizebuf > st.st_size)
                transmittedData += st.st_size;
        else
                transmittedData += sizebuf;
}
```

Na última trama não se envia informação nula.

Após este ciclo, manda um último packet de controlo e inicia o *llclose()*, terminando depois a execução.

5 Validação

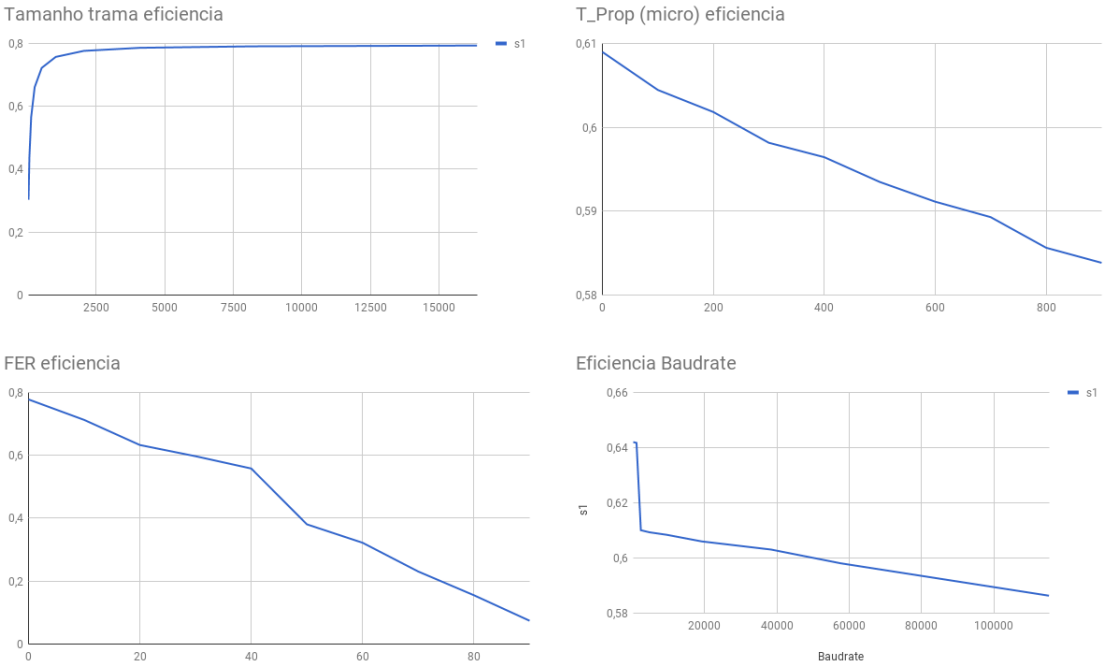
Para testar a fiabilidade do programa, sujeitamo-lo a dois tipos de testes: erros aleatórios durante a execução através de interferências na porta série, e interrupções deliberadas da porta.

Relativamente à primeira vertente, em todos os casos de erros, quer no cabeçalho da trama, quer no corpo de dados, o recetor enviava uma mensagem de rejeição, e o emissor reenviava a mensagem anterior.

Sobre as interrupções, caso se realizem antes, durante, ou após o envio, caso a interrupção exceda os três segundos, uma nova mensagem é enviada. Se o recetor processar duas vezes a mesma trama devido a isto, ou erros aleatórios induzidos, descarta os duplicados e manda o comando de mensagem recebida para o anterior.

6 Eficiência do Protocolo de Ligação de Dados

No protocolo de ligação, foi implementado um mecanismo de Automatic Repeat ReQuest para lidar com perdas de pacotes ou pacotes com erros nos dados. Desta forma, após ser enviada uma trama de informação o emissor espera por uma mensagem de confirmação por parte do recetor, requesitando ou a próxima trama, ou um pedido de retransmissão caso tenham sido encontrados erros. A eficiência do protocolo foi testada variando os valores da *Baudrate*, tamanho de trama, *Frame Error Ratio*, ou FER e Tempo de Propagação. Este último foi artificialmente simulando, colocando um *sleep* no *llwrite()* do emissor. As medidas resultantes permitiram a elaboração dos seguintes gráficos.



Em todos os gráficos divide-se a eficiência obtida pela teórica, sendo 1 o valor máximo. No primeiro gráfico varia-se o tamanho da trama, sendo o eixo X o valor em bytes. No do topo direito varia-se o tempo de propagação em microsegundos. No do canto inferior esquero, aumenta-se a FER, sendo esta percentual. O valor 20 significa que 20% das tramas recebidas têm erros. No último varia-se a *Baudrate*. A descida em eficiencia deve-se ao facto de se tratar de um ficheiro mais pequeno que o testado nos outros cenátrios.

7 Conclusões

Após o término do trabalho, é evidente ver a complexidade da criação dos protocolos de redes de computadores eficientes. Mesmo divindo esta tarefa por camadas, a eficiência observada demonstra que ainda é possível muita otimização para aumentar a *performance* do *software*. Para obter valores melhores, provavelmente teria de se adotar um sistema de comunicação diferente.

É possível observar o poder desta linguagem, que em cerca de mil linhas de código foi possível elaborar algo tão aparentemente complexo como este trabalho.