Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

UC Redes de Computadores

Protocolo de Ligação de Dados

Diogo Cadavez-up201603142

João Silva-up201604507

João Martins-up201708984

Introdução e Sumário

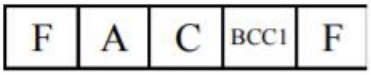
No âmbito da unidade curricular de Redes de Computadores foi nos proposto realizar um trabalho sobre transferência de dados entre computadores. Este serviu para estudarmos e compreendermos como se procede e realiza a ligação entre duas máquinas num nível baixo de programação. Houve duas camadas que tivemos de estabelecer, a primeira a camada de aplicação, que comunica com o utilizador, e a camada da ligação de dados, que é usada pelos computadores para comunicarem entre si. O objetivo era conseguir transferir ficheiros de um PC para o outro e também fazer o sentido inverso, pois a aplicação criada consegue fazer tanto de transmissor como de receptor. Os ficheiros escolhidos para os testes foram do tipo gif e do tipo txt devido ao seu tamanho reduzido que era mais prático. A ligação foi realizada através de portas série.

Estrutura do Código

Camada de ligação de dados

A camada de ligação de dados, representada neste trabalho pelos ficheiros linklayer.c e linklayer.h, esta camada faz a ligação entre a camada de aplicação e a API da porta série, permitindo assim o envio e recessão de frames tanto de dados como de controlo.

Frames de controlo:



F	**Flag**
A	**Campo de Endereço**
C	**Campo de Controlo** **SET (set up)** **DISC (disconnect)** **UA (unnumbered acknowledgment)** **RR (receiver ready / positive ACK)** **REJ (reject / negative ACK)**
BCC_1	**Campo de Proteção (cabeçalho)**

Frames de dados /informação:

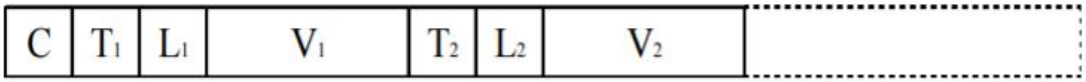
F	**Flag**
A	**Campo de Endereço**
C	**Campo de Controlo**
$D_1 \dots D_N$	**Campo de Informação (contém pacote gerado pela Aplicação)**
$BCC_{1,2}$	**Campos de Protecção independentes (1 – cabeçalho, 2 – dados)**

Os campos de proteção dos frames permitem verificar se o frame foi enviado/recebido sem qualquer erro.
As funções implementadas nesta camada foram, *llopen()*, *llwrite()* , *llread()* , *llclose()*,*receivecontrol()*,*sendcontrol()*, que serão posteriormente explicadas com maior detalhe.

Camada de Aplicação

A camada de aplicação representada pelos ficheiros cable.c e cable.h faz a ligação entre o utilizador, onde este tem a possibilidade de escolher o número da porta que pretende utilizar assim como se pretende enviar ou receber o ficheiro, e a camada de ligação de dados, invocando as funções que a constituem. Nesta camada são criados os pacotes de dados e de controlo do ficheiro que o utilizador pretende transmitir.

Pacotes de controlo:

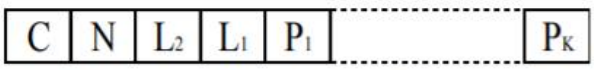


- C – campo de controlo (valores: 2 – *start*; 3 – *end*)
- Cada parâmetro (tamanho, nome do ficheiro, ou outro) é codificado na forma TLV (*Type*, *Length*, *Value*)
  - T (um octeto) – indica qual o parâmetro (0 – tamanho do ficheiro, 1 – nome do ficheiro, outros valores – a definir, se necessário)
  - L (um octeto) – indica o tamanho em octetos do campo V (valor do parâmetro)
  - V (número de octetos indicado em L) – valor do parâmetro

Estes pacotes marcam o início e o fim da transmissão de dados e transportam informação correspondente ao ficheiro a ser transmitido, permitindo ao receptor saber o nome e o tamanho do ficheiro que vai ser transmitido.

O pacote de controlo que marca o fim da transmissão, campo de controlo – END, permite ao receptor identificar o ficheiro que já foi recebido, isto no caso de serem transmitidos mais do que um ficheiro, que não consta no âmbito deste trabalho.

Pacotes de dados:



- C – campo de controlo (valor: 1 – dados)
- N – número de sequência (módulo 255)
- L_2 L_1 – indica o número de octetos (K) do campo de dados $(K = 256 * L_2 + L_1)$
- P_1 ... P_K – campo de dados do pacote (K octetos)

O valor de k é definido na camada de aplicação e representa o número de bytes do ficheiro que vão ser transmitidos em cada pacote de dados.

Os diferentes pacotes são criados no ficheiro apppck.c onde foram implementadas duas funções que permitem isso mesmo, uma para os pacotes de dados e outra para os de controlo.

Especificação das funções realizadas na camada de ligação

llopen(int port, int Tx_Rx)

Esta função recebe como argumentos, o valor que indica o numero da porta a ser utilizada (/dev/ttyS<port>), assim como um valor que indica se a função foi invocada pelo transmissor ou pelo receptor.
A função permite criar a porta a ser utilizada através da função *open ()*, obtendo o file descriptor. Estando esta etapa concluída, no caso do transmissor este envia um frame de controlo do tipo SET, recorrendo à função *sendcontrol (),* em seguida espera que o recetor responda com um frame de controlo do tipo UA, no recetor o processo é o inverso. Caso a ligação seja corretamente estabelecida a função dá retorno do file descriptor.

llwrite(int fd ,unsigned char *buffer , int length)

Recebe como argumentos o file descriptor da porta a ser utilizada assim como um pacote de dados ou de controlo, assim como o seu tamanho.
A função começa por calcular o BCC2 do pacote recebido, fazendo um xor entre cada um dos seus elementos. Em seguida é criado o frame a ser enviado, adicionando ao pacote recebido o cabeçalho e fazendo o o byte stuffing do mesmo.

Byte Stuffing

Este método verifica se algum dos bytes que vão ser enviados são iguais à flag que define o início e fim do frame (F = 0x7E) ou ao byte de escape (0x7D).
Caso isso se verifique esse byte é substituído pelo byte de escape seguido pelo resultado do xor entre 20 e o byte original.

- SE 0x7E obtém-se 0x7D 0x5E
- Se 0x7D obtém-se 0x7D 0x5D

Se o não fosse feito o byte stuffing corria-se o risco de o recetor não ler os dados na sua totalidade.

Completando estas etapas o transmissor envia o frame através da função *write()*.

Em seguida é implementado um ciclo de leitura que espera pela resposta do recetor, que permite saber se o frame foi recebido com sucesso ou se ocorreu erro na transmissão.

llread(int fd ,unsigned char *package)

Recebe como argumentos o file descritor da porta a ser utilizada e um apontador para um vetor onde os dados recebidos serão guardados , na camada de aplicação.
Na função foi implementado um ciclo de leitura que permite distinguir o cabeçalho do frame dos dados a serem recebidos.
Caso o cabeçalho seja recebido com sucesso e não tenham ocorrido erros, a função começa a extrair do frame os dados, fazendo simultaneamente o byte destuffing, só depois é que o BCC2 é novamente calculado e comparado com o recebido para determinar se o frame recebido não contem erros de transmissão.
Em caso de sucesso o recetor responde com um frame de controlo do tipo RR, confirmando ao transmissor que o frame foi bem recebido.
Caso contrário o recetor responde com REJ.

llclose(int fd , int Tx_Rx)

Recebe como argumento o file descritor da porta e um valor que indica se foi invocada pelo transmissor ou pelo recetor.
No caso do transmissor este envia um frame de controlo do tipo DISC e espera pela recessão do mesmo tipo de frame de controlo proveniente do recetor , em caso de sucesso envia um UA , e fecha a porta série através da função *close().*
No recetor este aguarda a recessão de um DISC para enviar o mesmo tipo de mensagem quando recebe o UA, termina também a ligação.

sendcontrol(int fd,char A,char C)

Esta função foi criada para enviar somente frames de controlo , esta recebe como argumentos o file descriptor da porta , assim como o campo de endereço e de controlo.

```c
int sendcontrol(int fd,char A,char C){

  int wr;
  unsigned char buffer[5];


  buffer[0] = F ;
  buffer[1] = A ;
  buffer[2] = C ;
  buffer[3] = buffer[1]^buffer[2]; //BCC
  buffer[4] = F ;

  wr = write(fd, buffer, 5);
  //printf(" wr:%d \n",wr);
  if(wr < 0) return -1;

  return 0;


}
```

receivecontrol(int port , char * c_frame)

Esta função tem como objetivo a recessão de frames de controlo , implementando assim um ciclo de leitura.

```c
int receivecontrol(int port , char * c_frame){

  int rd,i=0,end_stream = 0,S0=1,S1=0,S2=0,S3 = 0, end = 0;

  unsigned char buffer;

  printf("waiting for return frame:\n");
  while(end_stream == 0)
  {

    rd = read(port , &buffer , 1 );

    if(buffer == F && S0 == 1)
    {
      i=0;
      c_frame[i] = buffer;
      i++;
      S0 = 0;
      S1 = 1;
    }

    if(buffer != F && S1 == 1 && S0==0)
    {

      c_frame[i] = buffer;
      i++;

    }

    if (i==4 && buffer == F)
    {

      c_frame[i] = buffer;
      end_stream = 1;
      S1 = 0;

    }

  }

  return 1;

}
```

Especificação das funções realizadas na camada de aplicação

Nesta camada como foi dito anteriormente o utilizador tem a possibilidade de no mesmo programa escolher se pretende utilizar o transmissor ou o receptor.

Caso escolha o transmissor, é pedido ao utilizador que introduza o ficheiro que pretende transmitir, sendo este aberto em modo binário, o seu tamanho é calculado recorrendo a uma estrutura já existente na linguagem C.

```
3   struct stat buffer;
4
5   status = stat(filename , &buffer);
6   if(status == 0)
7   {
8       filesize = buffer.st_size;
9   }
```

As etapas seguintes passam por:

- Abrir a ligação com a porta, invocando *llopen()*;

```
85      fd = llopen(port , input);
```

- Criar o pacote de controlo START e envia-lo;

```
108         ncontrolpck = control_package(START , T1 ,a , fsize , T2 , b, filename , fileinfo);
109
110         wr = llwrite(fd , fileinfo , ncontrolpck);
111         if(wr < 0)
112         {
113             printf("error sending file info \n");
114             exit(1);
115         }
```

- O conteúdo do ficheiro que se pretende transferir é armazenado num vetor auxiliar, e os diferentes pacotes de dados são criados e enviados;

```
119         for(int i = 0 ; i < filesize ; i = i+frsize)
120         {
121 
122 
123             if(filesize-i > frsize) k = frsize;
124             else k = filesize-i;
125             memcpy(databuffer, filestr+i , k );
126 
127             //printf("valor do k : %d \n " , k);
128 
129             ncontrolpck = data_package( CDATA1 , (N++)%255 ,k, databuffer , package );
130 
131             package = realloc(package , ncontrolpck * sizeof(char));
132 
133             wr = llwrite(fd , package , ncontrolpck);
134 
135         }
```

- Depois de todos os pacotes serem enviados com sucesso o pacote de controlo END é enviado;

```
141         ncontrolpck = control_package(END , T1 , a , fsize , T2 , b , filename , fileinfo  );
142 
143         wr = llwrite(fd , fileinfo , ncontrolpck);
144         if(wr < 0)
145         {
146             printf("error sending file info \n");
147             exit(1);
148         }
```

- O ultimo passo é invocar a função llclose() , terminando assim a ligação;

No caso do recetor é implementado um ciclo de leitura que só acaba quando o frame de controlo END é recebido, este verifica qual o tipo de pacote que recebeu START , DATA ou END.

Caso seja um pacote de controlo START o recetor retira do pacote o nome do ficheiro e o seu tamanho, permitindo-lhe criar um novo ficheiro com esse nome.

```
190             if(package[0] == START)
191             {
192                 printf("recebi pacote de Start\n");
193 
194 
195                 if(package[1] == T1)
196                 {
197                     i=3;
198                     for( j = 0 ; j <= package[2]  ; j++)
199                     {
200                         char aux = package[i];
201                         if(j == 1)
202                         {
203 
204                             filesize = filesize + atoi(&aux)+1 ;
205                             printf("valor do filesize j==1 %d , valor do ciclo %d\n", filesize,package[2]-j);
206                             for(int ncasasdec = 1 ; ncasasdec < package[2]-j ; ncasasdec++)
207                             {
208                                 filesize = filesize*10;
209                             }
210                             break;
211                         }
212 
213                         filesize = (filesize + atoi(&aux));
214                         i++;
215                     }
216 
217                     i=10;
218                     filename = realloc(filename , package[10]*(sizeof(char)));
219                     for( k = 0 ; k< package[10] ; k++ )
220                     {
221 
222                         filename[k] = package[i];
223                         i++;
224 
225 
226                     }
227 
228                 }
```

```
269                     printf("vai criar novo ficheiro\n" );
270                     fptr = fopen(filename , "wb");
271                     if(fptr < 0 ) printf("failed to create file\n" );
272                     filestr = malloc(filesize*sizeof(char));
273 
```

Caso o pacote recebido seja de dados o recetor extrai o seu conteúdo e escreve diretamente no ficheiro criado.

```
284                 else if(package[0] == CDATA1)
285                 {
286                     printf("Recebi pacote de dados %d\n",package[1] );
287                     data_size = package[2]*256 + package[3];
288                                     i = 4;
289                     for(int d = 0 ; d < data_size ; d++)
290                     {
291                         realdata[d] = package[i++];
292                     }
293 
294                     fwrite( realdata, 1 , data_size , fptr);
295 
296 
297                 }
```

Quando recebe um pacote de controlo do tipo END o recetor sai do ciclo de leitura e invoca a função *llclose()*.

Conclusão

A realização deste trabalho permitiu-nos aprender e ficar a conhecer de forma mais aprofundada os protocolos, revelando a complexidade dos mesmos, que são necessários existir para que se possa fazer uma transferência de dados de confiança através do uso de portas série. Assim como todos os inconvenientes que por vezes podem surgir quando se trabalha com este tipo de portas.

Com ajuda do guião fornecido e recorrendo a pesquisas próprias conseguimos desenvolver o nosso próprio protocolo de ligação de dados, assim como a implementação da camada de aplicação, estas foram desenvolvidas tendo em conta a arquitetura em camadas, que se baseia no princípio de independência entre camadas.

Contudo durante a realização do projeto, surgiram algumas dificuldades, como por exemplo fazer debug ao código, tendo de se recorrer na sua maioria prints de controlo, de forma a permitir restringir a zona do código em que existia o bug, uma outra dificuldade sentida foi a sincronização do transmissor e do recetor que seria implementada recorrendo a time outs que faz com que o programa fique mais robusto em relação à ocorrência de erros ou de perda de ligação.

Contudo mesmo sem a presença de time outs conseguimos enviar o ficheiro pedido de um computador para o outro assim como outros ficheiros utilizados como teste.

É com satisfação que acabamos e cumprimos a grande maioria dos objetivos do trabalho. O esforço foi recompensado e no tempo estabelecido conseguimos alcançar o objetivo final do trabalho que era utilizar o protocolo apresentado para enviar e receber ficheiros com sucesso. Ficámos com vontade de aprimorar e desenvolver ainda mais todos este projeto e de aprender mais sobre tudo que advém de ligações entre computadores. Em última instância, reforço que este trabalho foi instrumental na nossa aprendizagem e todos os membros sentem que ficaram mais instruídos e mais preparados para situações futuras.