1º TRABALHO LABORATORIAL

Data Link Protocol

Mestrado em Engenharia Eletrotécnica e de Computadores
Redes de Computadores 2021/2022

Gonçalo Santos up202003537 1MEEC_T02
Guilherme Moreira up201806631 1MEEC_T02

11 de dezembro de 2021

Índice

Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular de Redes de Computadores visando implementar e analisar um protocolo de ligação de dados. Este protocolo, apelidado de *linklayer.c*, consiste na análise e processamento de dados em forma de tramas entre dois computadores através de uma porta série física ou virtual.

Neste relatório iremos analisar as funcionalidades implementadas de modo a explicar como as realizamos.

1. Parâmetros de configuração da ligação lógica

Primeiramente, foi desenvolvida uma função apelidada *llopen()* que conseguisse estabelecer uma conexão com a porta série com base na leitura e configuração dos parâmetros da ligação lógica indicados pelo utilizador na aplicação.

Nesta função é utilizada a *struct linklayer*, fornecida no ficheiro *linklayer.h* com os parâmetros enviados pela aplicação, nomeadamente o *baudrate*, o papel exigido ao protocolo, o número de tentativas de retransmissão (3), o *timeout* (3 segundos) e a porta série usada.

Por fim, estas definições são armazenadas na *struct termios*, sendo esta uma interface utilizada para controlar portas de comunicações assíncronas. Neste ponto é realizado uma verificação do *baudrate* final que a porta série vai possuir, recorrendo às funções *cfgetospeed()* do *termios.h* e *baudrate_check()* para certificar que os valores encontram – se uniformizados conforme o *standard* definido na *struct termios*, como apresentado no seguinte excerto de código.

```
newtio.c_cflag = baudrate_check(cfgetospeed(&oldtio)) | … ;

…

int baudrate_check(int baudrate_i)

{ int baudrate_f;

  switch (baudrate_i)

  { … case 9600: case 13:

      baudrate_f = B9600;//13 in decimal

      break;

    …

    case 38400: case 15:

      baudrate_f = B38400;//15 in decimal

      break; … }

…  }
```

2. Estabelecimento e encerramento da ligação lógica

Além do que foi referido anteriormente, a função *llopen()* vai efetuar a confirmação do estabelecimento da ligação lógica entre o transmissor e o recetor recorrendo a um conjunto específico de tramas de supervisão, de modo a conferir a estabilidade desta conexão.

De modo semelhante, este canal de comunicação é encerrado na função *llclose()* usando tramas de supervisão, restaurando as definições da porta série com a *struct termios* e culminando no fecho desta.

Na situação de um erro ou falha no estabelecimento/encerramento da ligação lógica o protocolo é encerrado e a aplicação sinalizada desse acontecimento.

O primeiro passo da ligação é dado pelo transmissor, que vai criar e enviar uma trama de 5 caracteres com a mensagem SET (*set-up*) para o recetor. Este vai aguardar uma trama de confirmação UA (*unumbered acknowledgment*) durante 3 segundos. Se for sucedido nesta tarefa, o protocolo vai informar a aplicação que pode passar à próxima fase.

Paralelamente, o recetor usa a função *statemachine()* para certificar que a trama recebida é de supervisão e que possui o *header* correto e esperado.

Em contraste, caso a resposta proveniente do recetor não for do tipo UA, o transmissor vai tentar retransmitir a trama SET durante 3 tentativas. Após a última, a aplicação é informada que não foi possível estabelecer a conexão via a porta série. De igual modo, se este falhar em receber algo de todo, a função *alarm()* vai desencadear um sinal após 3 segundos ativando o sistema de retransmissão. Estas ferramentas de *timeout* e tentativas de retransmissão são únicas ao transmissor encontrando – se ausentes no recetor.

A metodologia implementada no *llclose()* partilha semelhanças com o descrito acima. O transmissor vai tentar enviar uma trama de 5 caracteres com a mensagem DISC (*disconnect*), salvaguardada com o sistema de retransmissão, se este não receber outro DISC do recetor.

Este processo é finalizado com o transmissor a enviar uma trama UA. No entanto, o envio desta trama não é protegido pelo mecanismo de retransmissão, pois o transmissor não vai aguardar pela confirmação do recetor, tomando a iniciativa de informar a conclusão deste à aplicação.

3. Leitura das tramas (*framing*)

No transmissor é realizada a leitura das tramas de supervisão enquanto no recetor também são processadas as de informação.

Inicialmente, na configuração da porta série no *llopen()* são definidos o processamento *non-canonical* e um temporizador de 30 segundos entre cada carácter lido, caso o protocolo esteja a funcionar como transmissor. Caso esteja a funcionar como recetor, é escolhido um bloqueio de leitura de 5 caracteres, como apresentado no excerto de código abaixo.

```
if(ll->role == TRANSMITTER){

 newtio.c_cc[VTIME] = 30; // Inter-character timer unused

 newtio.c_cc[VMIN] = 0; // Blocking read until 5 chars
received }
```

```
else if(ll->role == RECEIVER) {

 newtio.c_cc[VTIME] = 0;  // Inter-character timer unused

 newtio.c_cc[VMIN]  =  5;//  Blocking  read  until  5  chars
received }
```

As tramas recebidas são sujeitas à máquina de estados implementada na função *statemachine()* que certifica que os caracteres recebidos e as suas respetivas posições correspondem com os da trama esperada. Para tal, esta função usa diversos *defines* com os valores hexadecimais dos caracteres esperados localizados no ficheiro *defines.h*, como mostrado no excerto de código abaixo.

```
#define FLAG 0X7E
```

Além disso, a máquina de estados recorre à variável *checksum* para avaliar se a trama passou com sucesso ou não nesta análise, invocando a função *error_check()* em caso de falhanço, ignorando se esta for espoletada no *header*.

Do lado do transmissor, para que o recetor consiga analisar corretamente os dados, na função *llwrite*, o *payload* é encapsulado para criar a trama I (informação) com *header* e *payload*, que à medida que é feito o seu encapsulamento é alvo de *stuffing*. Além disso, também é calculado o BCC2 para ser inserido na trama de informação.

Como o campo de controlo do *header* e o BCC1 das tramas de informação varia conforme o número de sequência, durante o encapsulamento é efetuada a verificação para ser enviada a trama com o *header* correto.

Por outro lado no recetor, a leitura das tramas de informação recebidas pela porta série é feita, quando a aplicação invoca a função *llread()*. Nesta função, é realizada a leitura das tramas recebidas pela porta série, através da variável global *fd* que contém o *file descriptor* que identifica esta.

Inicialmente, é "lido" o *header*, ou seja, os 4 primeiros caracteres da trama, com recurso à função *statemachine()*. Caso encontre algum erro lê a trama na sua totalidade, ou seja, até encontrar a última FLAG.

De seguida, é calculada a posição do BCC2 e retirado o BCC2 recebido, ao qual é feito uma verificação de *stuffing*,

Por último, procede-se à análise dos dados do ficheiro no *payload* que consiste: na verificação de *stuffing*, na passagem dos dados por uma máquina de estados e na avaliação do BCC2, ou seja, a comparação do BCC2 retirado da trama recebida com o BCC2 calculado na função *check_bcc2()*. (No capítulo 4 iremos explorar o que resulta da análise da trama recebida). A seguir, encontra – se o excerto de código referente à função *check_bcc2()*.

```
char check_bcc2(char buf[],int bufsize){char bcc2 = buf[0];

      for(int i = 1; i < bufsize; i++)

      {bcc2 ^= buf[i];}

      return bcc2;}
```

Aquando o envio das tramas de informação no *llwrite()*, é necessário verificar no *payload* da trama a existência do carácter FLAG (7E) com a função *stuffing()*. Se tal acontecer, esse carácter é substituído pela sequência de caracteres 7D-5E. Seguidamente, encontra – se o excerto de código referente à função *stuffing()*.

```
bool stuffing (char buf)

{ return buf == FLAG;}
```

Paralelamente, e como referido anteriormente, na receção de tramas de informação no *llread()*, o *payload* é analisado pela função *destuffing()*, que irá verificar a ocorrência da sequência de caracteres 7D-5E. Se esta função retornar verdadeiro, será efetuado o *destuffing*, sendo a sequência de caracteres substituída pelo carácter da FLAG (7E), revertendo o processo de *stuffing()* anterior. Continuamente, encontra – se o excerto de código referente à função de*stuffing()*.

```
bool destuffing(char buf1,char buf2)

{  return ((buf1 == 0x7d) && (buf2 == 0x5e)); }
```

4. Transmissão e receção de tramas

A transmissão das tramas de supervisão no estabelecimento e encerramento da ligação lógica são efetuadas nas funções *llopen()* e *llclose()* como referidas em pontos anteriores.

Equivalentemente, a transmissão das tramas de informação é realizada pelo transmissor quando a aplicação invoca a função *llwrite()*. Nesta função é realizado o encapsulamento dos dados, como referido anteriormente, o envio da trama e a receção da trama RR (*receiver ready/positive ACK*) enviada do lado do recetor.

A seguir, a trama criada é enviada, esperando 3 segundos pela resposta do recetor e, caso não receba uma trama RR, reenvia a trama de informação até que encontre ou uma trama RR, ou uma REJ (*reject/negative ACK*). Após 3 tentativas de envio, o transmissor procede ao encerramento da ligação lógica a partir da função *llclose()*.

Por último, a trama recebida é analisada e, caso seja uma trama RR, retorna o valor de *bytes* de dados enviados. Caso seja uma trama REJ, a função reenvia a trama de informação em que ocorreu o erro, recorrendo ao número de sequência passado pelo recetor na trama acolhida.

Paralelamente, do lado do recetor na função *llread()*, a trama é processada *byte* a *byte* enquanto que é analisada pela máquina de estados. No entanto, esta é escrutinada na sua totalidade para que o resto da trama não fique no *buffer* que existe na porta serie. De seguida, encontra – se um excerto de código referente à função *llread()* relativo à leitura *byte* a *byte*.

```
while (check != 2){ read(fd,tmp_buf,1);
      if(tmp_buf[0] == FLAG && bytes_read != 0)
      { frame_data[bytes_read] = tmp_buf[0];
        check = 2;
        break; }
      else
      {   frame_data[bytes_read] = tmp_buf[0];}
      bytes_read += 1;}
```

Entretanto, é verificado se é recebida uma trama DISC por parte do transmissor e, se tal acontecer, o recetor procede ao encerramento da ligação lógica a partir da função *llclose()*.

Na função *statemachine()* é realizado o cálculo do número de sequência de cada trama RR enviada para o transmissor de modo a este saber qual é a próxima trama a transmitir.

Como referido anteriormente, a trama recebida é analisada de modo que se retire o *payload*. O *payload*, é fundamentalmente o que o recetor recebe, ou seja, os dados do ficheiro, pelo que é armazenado no pacote passado pelo recetor quando invoca a função *llread()* na aplicação. Seguidamente, encontra-se um excerto de código da função onde se trata esta parte.

```
    for (int i = 0; i < bytes_read + 1; i++)
    { statemachine(frame_data[i],type);
      if((state == 4) && (i > 3) && (i < bcc2p)){
        if(destuffing(frame_data[i],frame_data[i+1])){
          destf_count++;
          packet[datasize] = FLAG;
          i++;
          datasize++;}
        else
        {  packet[datasize] = frame_data[i];
          datasize++;}
      }}
    return_check = datasize; // number of chars read
```

Finalmente, a função envia a trama RR como confirmação e retorna a quantidade de *bytes* do *payload*, obtido como referido no capítulo 3. Em caso de erro, ou seja, se o BCC2 não estiver correto, é enviado uma trama de rejeição REJ, ignorando o *payload*. Se acontecer algum erro no *header* toda a trama é ignorada para a aplicação reconhecer que o transmissor necessita de reenviar de novo a trama.

5. Trama REJ

A trama de rejeição é uma trama de supervisão e é implementada como tal. A sua maior diferença encontra-se no campo de controlo atualizado com o número de sequência relativo à trama onde ocorreu erro e consequentemente no BCC1, que resulta da operação XOR entre o endereço e campo de controlo.

Esta trama, como referido anteriormente, foi implementada no recetor na função *llread()*, sendo utilizada, caso haja algum erro nos dados recebidos.

6. Estatísticas da ligação

No protocolo implementado foi criada a função *protocol_stats()* para a análise da ligação estabelecida, caso esta seja solicitada pela aplicação. Estes dados são armazenados no decurso do protocolo na *struct protocol_data* que foi definida no *defines.h* como demostrado no excerto de código abaixo.

```
typedef struct protocol_data {

  int num_set_a;  // total number of SET frames sent

  int num_i_a;    // total number of I frames sent

  …

  int tries;      // total number of timeouts

} protocol_data;
```

Esta função irá apresentar no terminal as estatísticas sobre o número de tramas enviadas e recebidas, quer para o transmissor, como para o recetor, sendo que no primeiro, também será mostrado a quantidade de tramas retransmitidas e de *timeouts*. Estas tramas são distinguidas consoante a informação que lhes estão associadas (SET, I, DISC, UA, RR e REJ).

Além disso, também é apresentado o tempo total que o protocolo levou a cumprir o papel pretendido pela aplicação e o tempo do envio e da receção dos dados do ficheiro. Para efetuar os cálculos destes tempos foi usada a função *time()* da biblioteca *time.h*, tendo sido alcançado o tempo de 11 segundos (com uma resolução de 1 segundo) para um funcionamento sem erros com o ficheiro fornecido *penguin.gif*.

7. Gerador de erros aleatório

Infelizmente, não foi possível implementar este elemento de valorização, tendo sido dado prioridade à realização de testes, tanto com o *cable.c* como com a porta série física do laboratório, com o protocolo e à consolidação de funcionalidades previamente desenvolvidas.

Conclusão

Assim sendo, o protocolo de ligação lógica implementado cumpriu os objetivos principais deste trabalho laboratorial, tendo passado com sucesso os testes efetuados com o *cable.c* e com a porta série física, obtendo o resultado apresentado abaixo.