U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Protocolo de Ligação de Dados

(1º Trabalho Laboratorial – Redes de Computadores)

Fernando Barros - 201910223

Matilde Silva - 202007928

Sumário

Este relatório foi realizado no âmbito da unidade curricular Redes de Computadores (RCOM) do 1º semestre do 3º ano da Licenciatura em Engenharia Informática e de Computação (LEIC). O relatório tem como foco o primeiro trabalho laboratorial da UC, que tem como objetivo a transferência de dados através da implementação de um protocolo de comunicação entre duas máquinas. O trabalho prático foi dado como concluído com sucesso, uma vez que todos os objetivos definidos no início do mesmo, foram alcançados.

Introdução

O primeiro trabalho laboratorial visa a implementação de um protocolo de ligação de dados e a sua testagem com uma aplicação de transferência de ficheiros. O objetivo desta implementação é fornecer um serviço fiável de comunicação de dados entre dois sistemas conectados por um canal de transmissão neste caso, um cabo série.

O relatório vem complementar a parte prática do trabalho, de forma a explicar ao leitor a nossa implementação e os conceitos teóricos aplicados, tendo a estrutura seguinte:

- Arquitetura - Observação dos blocos funcionais e interfaces;
- Estrutura do código – Representação das APIs, principais estruturas de dados, principais funções e a sua relação com a arquitetura;
- Casos de uso principais – Identificação dos próprios e sequências de chamada de funções;
- Protocolos utilizados - Identificação dos principais aspetos funcionais bem como uma descrição da estratégia de implementação destes aspetos com apresentação de pedaços de código;
- Validação - Descrição dos testes efetuados com apresentação quantificada dos resultados;
- Eficiência do protocolo de ligação de dados - Caracterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido;
- Conclusão – Pequena síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

Arquitetura

Este projeto está dividido em duas camadas, de aplicação e de ligação. Na camada de ligação encontram-se as funções principais de todo o programa, é lá que estão as funções *open, write, read* e *close*. Estas garantem uma interface para que a camada de aplicação consiga usar estas funcionalidades. Nesta camada é feita a distinção entre a função de cada computador e o que cada um deles tem de fazer em cada momento da transferência de um ficheiro. Na camada de aplicação encontramos ainda funções que garantem que o ficheiro está a ser enviado e guardado corretamente.

Estrutura do código

O código encontra-se dividido em dois ficheiros principais: link_layer.c que tem como objetivo providenciar comunicação entre duas máquinas adjacentes uma à outra e garantir a integridade dos dados, e application_layer.c que trata da interação com os ficheiros e garante também a correta organização dos dados.

Temos ainda o ficheiro alarm.c que vem ajudar os dois ficheiros principais, permitindo um controle de erros quando a ligação entre os dois computadores (no momento de transferência de dados) é interrompida.

Link_layer

Funções principais

- **llopen() –** o emissor envia trama de supervisão *SET* e o recetor recebe trama *UA.*
- **llwrite() –** cria o *BBC* para o *dataPacket*, efetua *byte stuffing* das *I-frames*, cria uma nova *infoFrame* com o *dataPacket* (já *stuffed*) incluído, verifica se a *frame* recebida do *llread* tem erros e termina quando recebe uma mensagem de sucesso ou quando é excedido o limite de tentativas.
- **llread() –** lê a *pipe*, faz *destuffing* aos *bytes* lidos, verifica que os *BBCs* estão corretos, envia mensagem de confirmação de receção, positiva se correu tudo bem e negativa se ocorreu algum erro (ex: *BBC* incorreto).
- **llclose() –** o emissor envia trama de supervisão *DISC* ao recetor, que envia ao emissor também uma trama *DISC* e posteriormente recebe uma trama *UA.*

Variáveis globais

```
8   extern int alarmEnabled, alarmCount;
9   int senderNumber = 0, receiverNumber = 1;
10  int nTries, timeout, fd, lastFrameNumber = -1;
```

Estruturas de dados

```
34  enum state {
35      STATE0,
36      STATE1,
37      STATE2,
38      STATE3,
39      STATE4,
40      STATE5
41  } typedef STATE;
```

```
18  typedef enum
19  {
20      LlTx, //transmissor
21      LlRx, //recetor
22  } LinkLayerRole;
23
24  typedef struct
25  {
26      char serialPort[50];
27      LinkLayerRole role;
28      int baudRate;
29      int nRetransmissions;
30      int timeout;
31  } LinkLayer;
```

Application_layer

Funções principais

- **getControlPacket() –** constrói um pacote de controlo, para ser enviado numa *I-frame*
- **getDataPacket() –** constrói um pacote de dados do ficheiro, para ser enviado numa *I-frame*
- **applicationLayer() –** chama as funções que permitem que em cada máquina seja executada a devida tarefa

Alarm

Funções

- **alarmHandler() –** função chamada sempre que o alarme emite uma interrupção, incrementa alarmCount e desativa o alarme.
- **startAlarm() –** função que inicia o alarme

Casos de uso principais

Interface

A interface permite que o transmissor selecione um arquivo para enviar. O usuário usa a consola para executar o programa, dando um conjunto de argumentos. Do lado do emissor, deve ser inserida a porta de série a ser utilizada (ex. **/dev/ttyS0**) e o nome do arquivo a ser enviado (ex. **pinguim.gif**). Do lado do recetor, apenas deve ser inserida a porta de série.

Transmissão de dados

A transmissão dos dados ocorre através de uma porta de série, entre dois computadores, seguindo a sequência de eventos:

- O emissor escolhe o ficheiro a ser enviado;
- É configurada a ligação entre os dois computadores;
- É estabelecida a ligação;
- O transmissor faz o envio dos dados trama a trama;
- Simultaneamente, o recetor recebe os dados, também trama a trama;
- Mensagens quanto ao estado da ligação e fase de execução vão aparecendo na consola para que o utilizador consiga perceber o que está a ocorrer;
- Conforme se vai dando a receção dos dados, o recetor guarda-os num ficheiro com o mesmo nome do ficheiro enviado pelo transmissor;
- A ligação é terminada.

Protocolos utilizados

• Protocolo de ligação de dados

No protocolo de ligação de dados foram implementadas as funções *llopen, llwrite, llread* e llclose, tal como é pedido no guião do trabalho laboratorial. Estas são as principais funções do protocolo, e fornecem uma interface para que o protocolo de aplicação consiga utilizar as funcionalidades.

Estas funções fazem, respetivamente: o estabelecimento da ligação entre o emissor e recetor, o envio de uma trama de informação (finaliza apenas quando este é terminado com sucesso ou quanto é ultrapassado o número de tentativas da transferência), a receção de uma trama de informação (termina da mesma forma que a função anterior), e o término da ligação entre os dois computadores (emissor e recetor). As leituras de uma qualquer trama são feitas através de uma *state machine*, que vai recebendo byte a byte o arquivo, executando mudanças de estado.

Implementação do loop principal no lado do emissor na função do llopen:

```
while(alarmCount < nTries){

    if(!alarmEnabled){
        int bytes = write(fd, buf, sizeof(buf));
        printf("\nSET message sent, %d bytes written\n", bytes);
        startAlarm(timeout);
    }

    int result = read(fd, parcels, 5);
    if(result != -1 && parcels != 0 && parcels[0]==0x7E){
        //se o UA estiver errado
        if(parcels[2] != 0x07 || (parcels[3] != (parcels[1]^parcels[2]))){
            printf("\nUA not correct: 0x%02x%02x%02x%02x%02x\n", parcels[0], parcels[1], parcels[2], parcels[3], parcels[4]);
            alarmEnabled = FALSE;
            continue;
        }

        else{
            printf("\nUA correctly received: 0x%02x%02x%02x%02x%02x\n", parcels[0], parcels[1], parcels[2], parcels[3], parcels[4]);
            alarmEnabled = FALSE;
            break;
        }
    }

}

if(alarmCount >= nTries){
    printf("\nAlarm limit reached, SET message not sent\n");
    return -1;
}
```

Enquanto o loop está a correr, o programa lê 5 bytes da pipe e verifica a correção dos mesmos. Sempre que o alarme se desligar, voltamos a escrever na pipe a SET message e re-ativamos o alarme.

Implementação do loop principal no lado do recetor na função do llopen:

```
120     else
121     {
122         unsigned char buf[1] = {0}, parcels[5] = {0}; // +1: Save space for the final '\0' char
123
124         STATE st = STATE0;
125         unsigned char readByte = TRUE;
126
127         // Loop for input
128         while (STOP == FALSE)
129         {
130             if(readByte){
131                 int bytes = read(fd, buf, 1); //ler byte a byte
132                 if(bytes==0 || bytes == -1) continue;
133             }
134
135
136 >           switch (st) …
204         }
205
206         parcels[2] = 0x07;
207         parcels[3] = parcels[1]^parcels[2];
208
209         //preciso de estar dentro da state machine ate receber um sinal a dizer que o UA foi corretamente recebido
210
211         int bytes = write(fd, parcels, sizeof(parcels));
212         printf("UA message sent, %d bytes written\n", bytes);
213     }
```

Enquanto o loop decorre, o recetor lê a pipe e mediante o seu valor é alterado o estado na state machine (código ocultado). Aquando da mudança de estados, verificamos se os bytes lidos correspondem à SET message. A state machine só termina quando recebe uma SET message completa e correta.

A função llwrite encarrega-se da criação da information frame, BCC2 e byte stuffing. Após completos estes passos, escreve na pipe a information frame criada. O loop utilizado para o envio da frame é semelhante ao do llopen, no entanto, a mensagem de resposta esperada é a RR ao oposto de um UA.

```
for(int i=0; i<bufSize; i++){
    BCC = (BCC ^ buf[i]);
}

infoFrame[0]=0x7E; //Flag
infoFrame[1]=0x03; //Address
infoFrame[2]=(senderNumber << 6); //Control
infoFrame[3]=infoFrame[1]^infoFrame[2];
```

Criação do BCC2 e preparação da information frame.

Por sua vez, llread começa por usar uma state machine para ler os bytes da pipe, uma a um. Enquanto os lê, verifica apenas se não recebe dois FLAG bytes seguidos, garantindo assim a leitura de uma só information frame, ao contrário de pedaços de várias. Após receber uma information frame completa, faz o de-stuffing e verifica os valores de controlo, BCC1 e BCC2, e mediante os resultados, escreve uma resposta de RR ou REJ.

```
if(packet[size-2] == BCC2){

    if(packet[4]==0x01){
        if(infoFrame[5] == lastFrameNumber){
            printf("\nInfoFrame received correctly. Repeated Frame. Sending RR.\n");
            supFrame[2] = (receiverNumber << 7) | 0x05;
            supFrame[3] = supFrame[1] ^ supFrame[2];
            write(fd, supFrame, 5);
            return -1;
        }
        else{
            lastFrameNumber = infoFrame[5];
        }
    }
    printf("\nInfoFrame received correctly. Sending RR.\n");
    supFrame[2] = (receiverNumber << 7) | 0x05;
    supFrame[3] = supFrame[1] ^ supFrame[2];
    write(fd, supFrame, 5);
}

else {
    printf("\nInfoFrame not received correctly. Error in data packet. Sending REJ.\n");
    supFrame[2] = (receiverNumber << 7) | 0x01;
    supFrame[3] = supFrame[1] ^ supFrame[2];
    write(fd, supFrame, 5);

    return -1;
}
```

Por fim, a função llclose, do lado do recetor primeiro espera a leitura de uma mensagem DISC, após qual emprega o uso de um loop igual ao de llopen, já que espera a receção da mensagem UA. No lado do transmissor, primeiro é usado um loop semelhante ao do llopen, começando por enviar uma mensagem DISC. Porém, em vez de a mensagem de resposta esperada ser um UA, passa a ser um DISC, sendo que, após confirmada a receção de tal é enviada uma só mensagem UA e a função llopen termina.

- **Protocolo de aplicação de dados**

No protocolo de ligação de dados foram implementadas as funções *getControlPacket, getDataPacket* e *applicationLayer.* As duas primeiras permitem a construção de um pacote de controlo e de um pacote de dados do ficheiro, respetivamente, para ser enviado numa *I-frame*. A última faz o controlo da tarefa a ser realizada por cada computador a cada instante.

```
while(fileNotOver){

    //comeco por ler a file stream
    //se deixar de haver coisas para ler corro este codigo
    if(!fread(&curByte, (size_t)1, (size_t) 1, fileptr)){
        fileNotOver = 0;
        sizePacket = getDataPacket(bytes, &packet, nSequence++, index);

        if(llwrite(packet, sizePacket) == -1){
            return;
        }
    }

    //se o valor de index for igual a nBytes, significa que o ja passamos por nByte elementos
    else if(nBytes == index) {
        sizePacket = getDataPacket(bytes, &packet, nSequence++, index);

        if(llwrite(packet, sizePacket) == -1){
            return;
        }

        memset(bytes,0,sizeof(bytes));
        memset(packet,0,sizeof(packet));
        index = 0;
    }

    bytes[index++] = curByte;
}

fclose(fileptr);
```

Loop, no lado do transmissor, que lê o ficheiro pretendido, byte a byte, e chama continuamente a função llwrite até terminada a leitura.

```
while(readBytes){

    unsigned char packet[600] = {0};
    int sizeOfPacket = 0, index = 0;

    if(llread(&packet, &sizeOfPacket)==-1){
        continue;
    }


    if(packet[0] == 0x03){
        printf("\nClosed penguin\n");
        fclose(fileptr);
        readBytes = 0;
    }
    else if(packet[0]==0x02){
        printf("\nOpened penguin\n");
        fileptr = fopen(filename, "wb");
    }
    else{
        for(int i=4; i<sizeOfPacket; i++){
            fputc(packet[i], fileptr);
        }
    }
}
```

Loop, no lado do recetor, que recebe o data packet da função llread e de acordo com o valor do controlo, abre, fecha ou escreve para o novo ficheiro.

Validação e eficiência do protocolo de ligação de dados

Em condições ideais, sem ruído nem interferências, um ficheiro de 10968 bytes demora aproximadamente 8 segundos a transmitir, BAUDRATE = 38400. De realçar, qualquer uma information frame contém no máximo 200 bytes do ficheiro a transmitir (sem stuffing). Como tal, o valor efetivo para a transferência de bytes ronda os 1371 bytes/segundo.

Testes ao código para ficheiros superiores a 1MB não tiveram sucesso na transferência. A certa altura, o lado do emissor rejeitava toda a informação lida, efetivamente falhando a transimissão. Não foi possível determinar a exata causa deste erro.

Porém, no que toca a interrupções da transmissão, o protocolo Stop-And-Wait, observou-se como extremamente eficaz, porquanto, sempre que o recetor falha em ler qualquer informação da pipe, volta a tentar a leitura na próxima iteração, parando apenas quando recebe uma information frame completa (information frame, nesta instância, é definida como uma sequência de dados, com comprimento superior a 2, que começa e termina com 0x7E). Durante a verificação, no entanto, é provável que os dados lidos sejam rejeitados, pelo que adiciona um overhead, o tempo da re-transmissão da frame adicionado ao tempo que a porta de série esteve desconectada.

Conclusão

Este projeto consiste na implementação de um serviço de comunicação entre dois computadores ligados por uma porta de série, implementando um protocolo de ligação de dados fiável.

Com a elaboração deste trabalho, foi-nos permitida uma compreensão e reflexão quanto ao protocolo de ligação de dados, mais especificamente quanto à estrutura das tramas, o processo de encapsulamento e o envio e receção de informação.

Anexo (código fonte)

```
switch (st)
{
case STATE0:
    //state0 porque a primeira coisa que quero rec
    if(buf[0] == 0x7E){
        st = STATE1;
        infoFrame[sizeInfo++] = buf[0];
    }
    //se nao receber a flag significa que estou a
    break;

case STATE1:
    //no STATE1 eu quero receber qualquer coisa ex
    // ou seja, EU NAO QUERO EM QUALQUER CASO LER
    //por isso este estado serve só para receber u
    if(buf[0] != 0x7E){

        st = STATE2;
        infoFrame[sizeInfo++] = buf[0];
    }
    //Se eu ler duas FLAGS seguidas sei que a flag
    else{
        memset(infoFrame, 0, 600);
        st = STATE1;
        sizeInfo = 0;
        infoFrame[sizeInfo++] = buf[0];
    }
    break;

case STATE2:
    //no state2 eu ja garanti que estou a ler uma
    if(buf[0] != 0x7E){
        infoFrame[sizeInfo++] = buf[0];
    }
    //quando receber a flag significa que a trama
    else if(buf[0] == 0x7E){

        STOP = TRUE;
        infoFrame[sizeInfo++] = buf[0];
        readByte = FALSE;
    }
    break;

default:
    break;
}
```

State machine usada na função llread

```
switch (st)
{
case STATE0:
    if(buf[0] == 0x7E){
        st = STATE1;
        parcels[0] = buf[0];
    }
    break;

case STATE1:
    if(buf[0] != 0x7E){
        st = STATE2;
        parcels[1] = buf[0];
    }
    else {
        st = STATE0;
        memset(parcels, 0, 5);
    }
    break;

case STATE2:
    if(buf[0] != 0x7E){
        st = STATE3;
        parcels[2] = buf[0];
    }
    else {
        st = STATE0;
        memset(parcels, 0, 5);
    }
    break;
```

State machine usada na função llopen (1/2)

```
case STATE3:
    if(buf[0] != 0x7E){
        parcels[3] = buf[0];
        st = STATE4;
    }
    else {
        st = STATE0;
        memset(parcels, 0, 5);
    }
    break;

case STATE4:
    if(buf[0] == 0x7E){
        parcels[4] = buf[0];
        st = STATE5;
        readByte = FALSE;
    }

    else {
        st = STATE0;
        memset(parcels, 0, 5);
    }
    break;
case STATE5:
    if(((parcels[1])^(parcels[2]))==(parcels[3])){
        printf("\nGreat success! SET message received without errors\n\n");
        STOP = TRUE;
    }
    else {
        st = STATE0;
        memset(parcels, 0, 5);
        readByte = TRUE;
    }
    break;
default:
    break;
}
```

State machine usada na função llopen (2/2)

```
for(int i=0; i<bufSize; i++){
    if(buf[i]==0x7E){
        infoFrame[index++]=0x7D;
        infoFrame[index++]=0x5e;
        continue;
    }
    else if(buf[i]==0x7D){
        infoFrame[index++]=0x7D;
        infoFrame[index++]=0x5D;
        continue;
    }

    infoFrame[index++]=buf[i];
}

if(BCC == 0x7E){
    infoFrame[index++]=0x7D;
    infoFrame[index++]=0x5e;
}

else if(BCC == 0x7D){
    infoFrame[index++]=0x7D;
    infoFrame[index++]=0x5D;
}

else {infoFrame[index++]=BCC;}
```

Byte stuffing na função llwrite

```
for(int i=0; i<sizeInfo; i++){
    if(infoFrame[i] == 0x7D && infoFrame[i+1]==0x5e){
        packet[index++] = 0x7E;
        i++;
    }

    else if(infoFrame[i] == 0x7D && infoFrame[i+1]==0x5d){
        packet[index++] = 0x7D;
        i++;
    }

    else {packet[index++] = infoFrame[i];}
}
```

Byte de-stuffing na função llread

```
if(strlen(filename) > 255){
    printf("size of filename couldn't fit in one byte: %d\n",2);
    return -1;
}

unsigned char hex_string[20];

struct stat file;
stat(filename, &file);
sprintf(hex_string, "%02lX", file.st_size);

int index = 3, fileSizeBytes = strlen(hex_string) / 2, fileSize = file.st_size;

printf("\nfilesize: %d\n hex_string: %s", file.st_size, hex_string);

if(fileSizeBytes > 256){
    printf("size of file couldn't fit in one byte\n");
    return -1;
}

if(start){
    packet[0] = 0x02;
}
else{
    packet[0] = 0x03;
}
```

Criação do control packet, função getControlPacket. (1/2)

```
  packet[1] = 0x00; // 0 = tamanho do ficheiro
  packet[2] = fileSizeBytes;


  for(int i=(fileSizeBytes-1); i>-1; i--){
      packet[index++] = fileSize >> (8*i);
  }


  packet[index++] = 0x01;
  packet[index++] = strlen(filename);

  for(int i=0; i<strlen(filename); i++){
      packet[index++] = filename[i];
  }

  sizeOfPacket = index;

  return sizeOfPacket;
```

Criação do control packet, função getControlPacket. (2/2)

```
/*Funcao testada e funcional*/
int getDataPacket(unsigned char* bytes, unsigned char* packet, int nSequence, int nBytes){

    int l2 = div(nBytes, 256).quot , l1 = div(nBytes, 256).rem;

    packet[0] = 0x01;
    packet[1] = div(nSequence, 255).rem;
    packet[2] = l2;
    packet[3] = l1;

    for(int i=0; i<nBytes; i++){
        packet[i+4] = bytes[i];
    }

    return (nBytes+4); //tamanho do data packet

}
```

Criação do data packet, função getDataPacket.

Link para o repositório GitHub: https://github.com/matilde-silva-21/RCOM