Protocolo de Ligação de Dados

Relatório do 1º Trabalho Laboratorial

Mestrado integrado em Engenharia Informática e Computação

Redes de Computadores

Bruno Marques - 201405781

João Loureiro - 201405652

José Cruz - 201403526

José Costa - 201402717

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

3 de novembro de 2016

Índice

Sumário 3
1. Introdução 3
2. Arquitetura 3
3. Estrutura do Código 4
4. Casos de Uso Principais 5
   4.1. Sequência de chamada de funções 5
5. Protocolo de Ligação Lógica 5
   5.1. Principais aspetos funcionais 6
   5.2. Estratégia de implementação 6
      5.2.1. send_cycle 6
      5.2.2. ll_open e ll_close 7
      5.2.3. ll_write e ll_read 7
6. Protocolo de Aplicação 8
   Pacotes de controlo 8
   Pacotes de informação 8
   Envio e Recepção 9
6. Validação 9
7. Elementos de Valorização 9
8. Conclusões 10

Sumário

O presente relatório serve de apoio ao 1º projeto realizado no âmbito da Unidade Curricular "Redes de Computadores" do 3º ano do Mestrado integrado em Engenharia Informática e Computação. Projeto este intitulado "Protocolo de Ligação de Dados" e cujo objetivo era transferir ficheiros entre dois computadores usando a porta de série tendo em conta os conteúdos abordados nas aulas teóricas e práticas, dos quais se destacam *Application Layer, Data Link Layer* e *Physical Layer.*

1. Introdução

O trabalho prático em questão visa a implementação de um protocolo de ligação de dados, de acordo com a especificação do guião, e o teste deste mesmo protocolo com uma aplicação simples de transferência de ficheiros, igualmente especificada. Este relatório serve para especificar o projeto tanto do ponto de vista prático como teórico e serão caracterizadas todas as funcionalidades implementadas no projeto. Todo o projeto foi desenvolvido em ambiente Linux utilizando a linguagem C e portas de série RS-232, com comunicação assíncrona.

O relatório está dividido em diferentes secções, sendo estas:

- **Arquitetura:** Especificação dos Blocos Funcionais e da Interface.
- **Estrutura do Código:** Descrição das principais APIs e Estruturas de Dados utilizadas, bem como as principais Funções e a sua relação com a Arquitetura.
- **Casos de Uso Principais:** Fazer a sua Identificação e abordar as sequencias de chamada de funções.
- **Protocolo de Ligação Lógica:** Identificação dos principais aspetos funcionais da *LinkLayer*, descrevendo a estratégia da sua implementação.
- **Protocolo de Aplicação:** Identificação dos principais aspetos funcionais da *ApplicationLayer*, descrevendo a estratégia da sua implementação.
- **Validação:** Descrição dos testes efetuados ao programa com apresentação quantificada dos resultados.
- **Elementos de Valorização:** Identificação dos elementos de valorização implementados e descrição da estratégia da sua implementação.

2. Arquitetura

O projeto está dividido em duas camadas funcionais: a camada do protocolo de ligação de dados e a camada de aplicação (implementados em diferentes ficheiros source e header, respetivamente *.c e*.h). Os ficheiros link.c e link.h representam a camada do protocolo de ligação de dados e os ficheiros application.c e application.h representam a camada de aplicação. A camada do protocolo de ligação de dados tem as funções de sincronismo e abertura, fecho e configuração da porta de série, *stuffing* e *unstuffing* de bytes. A camada de aplicação cria as tramas e é responsável pelo envio e receção do ficheiro.

A interface está implementada em interface.c e interface.h onde o utilizador começa por selecionar se é o recetor ou o emissor e permite a escolha de valores de alguns parâmetros referentes à transferência do ficheiro referindo também os limites destes mesmos valores, sendo estes: o *Baudrate* , que porta utilizar(se /dev/ttyS0 ou /dev/ttyS1), o tamanho máximo do campo de informação das tramas, o intervalo *Timeout*, o número máximo de retransmissões e o nome do ficheiro a ser enviado.

3. Estrutura do Código

Ambas as camadas são representadas por uma estrutura de dados, estruturas estas que estão implementadas em utils.h.

A estrutura da camada do protocolo de ligação de dados, bem como as suas principais funções estão abaixo.

```
typedef struct {
        char port[20]; /*Dispositivo /dev/ttySx, x = 0, 1*/
        int baudRate;   /*Velocidade de transmissão */
        int sequenceNumber;   /*Número de sequência da trama: 0, 1*/
        int timeout;    /*Valor do temporizador: 1 s*/
        int numTransmissions; /*Número de tentativas em caso de falha*/
        unsigned char frame[MAX_SIZE];  /*Trama*/
} LinkLayer;
```

```
void alarmHandler();
int badSET(unsigned char* set);
int badUA(unsigned char *ua);
int badDisc(unsigned char *disc);
int byteStuff(Array* inArray, Array* outArray);
int byteUnstuff(Array* inArray, Array* outArray);
int llopen(ApplicationLayer* appL, LinkLayer* linkL, struct termios* oldtio, Stats* stats);
int llclose(ApplicationLayer* appL, struct termios* oldtio, Stats* stats);
int llwrite(int fd, unsigned char* packet, size_t packetLength, LinkLayer* linkL, Stats* stats);
int llread(int fd, unsigned char* packet, size_t* packetLength, LinkLayer* linkL, Stats* stats);
```

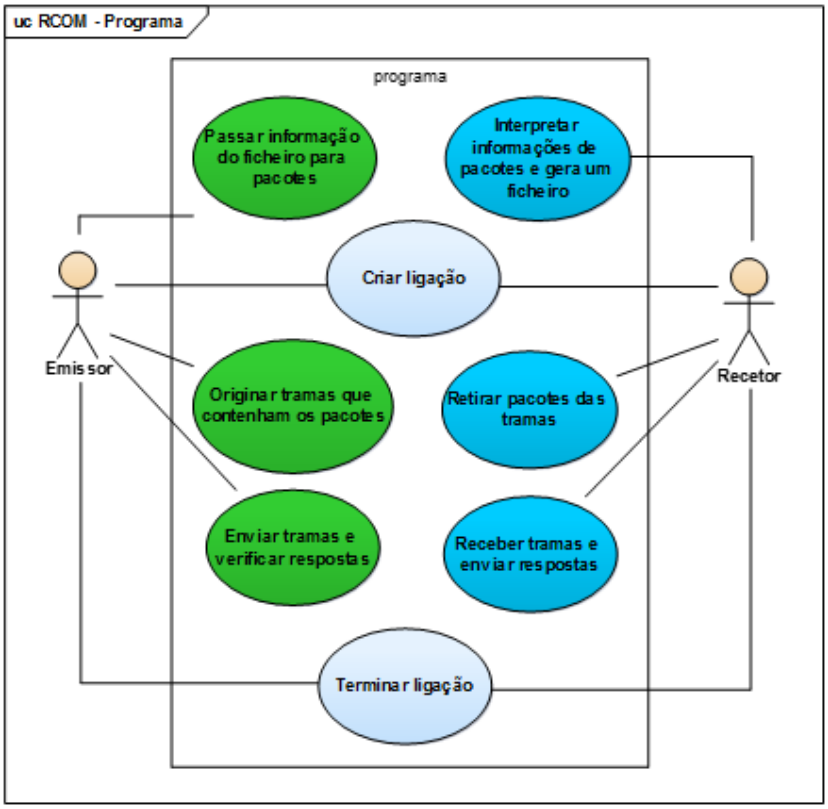
A estrutura da camada de aplicação e as suas principais funções.

```
typedef struct {
        int fileDescriptor; /*Descritor correspondente à porta série*/
        int status;     /*TRANSMITTER | RECEIVER*/
} ApplicationLayer;
```

```
void loadFile(char * filename);
void receiveFile();
int writeToFile(unsigned char * trama, int file_d, int * n_trama);
void createControlPackage(unsigned char * pack, long int size,  char * filename, int valor);
void createDataPackage(unsigned char * pack, unsigned char * info_trama, int n_trama, long int size_trama);
```

Para além dos ficheiros onde as camadas estão implementadas (link.c, link.h,application.c e application.h), existem ainda outros ficheiros: utils.c e utils.h (onde estão definidas as estruturas de dados das duas camadas e algumas funções auxiliares, por exemplo, de manipulação de arrays) e interface.c e interface.h(onde está definida a interface).

4. Casos de Uso Principais



4.1. Sequência de chamada de funções

Em traços gerais, o nosso programa segue a seguinte sequência de funções durante a sua execução:

1. Obter a partir do utilizador as definições a usar na conexão (`menu_cycle()`).
2. Criar e abrir a conexão partilhada, tanto no emissor como no recetor (`ll_open()`).
3. No emissor:
    a. Ler a informação do ficheiro escolhido para transmissão e empacotá-la (`loadFile()`).
    b. Criar e enviar tramas que contenham os pacotes de informação, decidindo o próximo passo pela resposta do emissor (`ll_write()`).
4. No recetor:
    a. Receber tramas de informação e enviar uma resposta ao emissor, retirando das tramas os pacotes de informação (`ll_read()`).
    b. Gerar um novo ficheiro a partir dos bytes dos pacotes recebidos (`receiveFile()`).
5. Terminar a ligação (`ll_close()`).

5. Protocolo de Ligação Lógica

O protocolo de ligação lógica é definido e implementado no ficheiro *link.c*, que inclui as funções que o protocolo de aplicação utiliza para enviar informações de um controlador para o outro.

5.1. Principais aspetos funcionais

- Definir a configuração da porta série.
- Criar a ligação entre portas série.
- Usar o byte *stuffing* e *destuffing* para criar tramas.
- Enviar e receber tramas de informação ou de controlo.
- Terminar a ligação, repondo as configurações prévias da porta-série.

5.2. Estratégia de implementação

5.2.1. send_cycle

A transmissão de informação pela porta série segue um padrão de envio de comando e receção de resposta. Assim, foi elaborada uma função que permite escrever um determinado comando para um descritor de ficheiro (que no presente caso é a porta-série) e que reenvia esse comando um determinado número de vezes, utilizando o alarme, até receber resposta por esse mesmo descritor. As funções ll_open, ll_close, ll_read e ll_read recorrem todas ao send_cycle quando se pretende trocar informação entre controladores.

```
int send_cycle(int fd, unsigned char * sendMsg, int size, unsigned char * received,
Stats* stats){

    flag = 1;
    numOfTries = 0;
    int res;

    (void)signal(SIGALRM, alarmHandler); /* sets alarmHandler function as SIGALRM
handler*/

    int writtenChars = 0;

    //Cycle that sends the SET bytes, while waiting for UA
    while(numOfTries < max_num_tries){
        if(flag)
        {
            alarm(time_out); /* waits x seconds, then activates a SIGALRM */
            flag = 0; /* doesn't resend a signal until an alarm is handled */

            tcflush(fd, TCIOFLUSH);

            writtenChars = write(fd, sendMsg, size); // writes the flags
        }

        res = read(fd, received, size);

        if(res >= 1)
        {
            return writtenChars;
        }
        if(numOfTries == max_num_tries)
        {
            stats->numTimeouts++;
        }
    }
    return -1;
}
```

5.2.2. ll_open e ll_close

A função **ll_open** configura a porta-série a partir dos atributos da struct *applicationLayer*, que foi definida para o programa. Do lado do emissor é enviado o comando SET, através da função send_set, até ser recebida a resposta UA vinda do emissor.

```
if(appL->status == TRANSMITTER)
        send_set(fd, stats);
else
        receive_set(fd);
```

A função **ll_close** envia o comando DISC a partir do emissor, quando os pacotes de informação acabam de ser transmitidos. O recetor, ao receber o comando de terminação, responde também com uma mensagem DISC, que, ao ser interpretada pelo emissor, é retornada com UA.

```
if(appL->status == TRANSMITTER)
        close_set(fd, stats);
else
        close_ua(fd);
```

5.2.3. ll_write e ll_read

Como as funções ll_write e ll_read necessitam de alocação dinâmica, foi elaborada uma struct *Array* que possui funções capazes de simular um vetor dinâmico em C, de forma a não ser necessário lidar sempre com alocações de memória. O byte *stuff* e *destuff* é feito recorrendo a funções desta struct, como por exemplo:

```
void insertArray(Array *a, unsigned char element) {
  if (a->used == a->size) {
    a->size *= 2;   // double size if full
    a->array = (unsigned char*)realloc(a->array, a->size * sizeof(unsigned char));
  }
  a->array[a->used++] = element;
}
```

A função **ll_write** é executada pelo transmissor e começa por inicializar a trama que se vai enviar com os bytes de controlo do cabeçalho, incluindo o *BCC1*. De seguida, pega no pacote que vai enviar e calcula o *BCC* dos bytes do pacote, anexando o BCC à informação de envio. O *Array* resultante passa depois por byte *stuffing* e recebe o cabeçalho e a FLAG final. O conjunto de bytes obtido depois deste processo é enviado através da send_cycle para o recetor.

```
Array packetArray;
initArray(&packetArray, 1);

copyArray(packet, &packetArray, packetLength);

Array stuffedArray;
initArray(&stuffedArray, 1);

initializeInformationFrame(&stuffedArray, linkL);

addBodyBCC(&packetArray);

byteStuff(&packetArray, &stuffedArray);

endInformationFrame(&stuffedArray);
```

A função **ll_open** tem o mesmo funcionamento, mas com o sentido contrário ao da ll_write, fazendo uso de uma função chamada `generateResponse()` para verificar a validade da trama que recebeu e enviar a devida resposta de volta ao transmissor.

6. Protocolo de Aplicação

O protocolo de aplicação é definido e implementado no ficheiro *application.c*, onde se pode visualizar neste as funções responsáveis por enviar e receber um determinado ficheiro e de criação de pacotes de controlo e de informação.

Os pacotes de controlo vão ser utilizados nas funções de envio e de recepção, sendo que a primeira é responsável por criar e enviar, a segunda por receber e ler.

Os processos de envio/aceitação são executados após o ll_open, e depois de terminarem, vai seguir para o ll_close.

Pacotes de controlo

Este tipo de pacotes tem como finalidade delimitadora do envio do ficheiro, ou seja, servem para indicar o início e fim do processo de envio, bem como passam informação acerca

```
void createControlPackage(unsigned char * pack, long int size,  char * filename, int valor)
```

deste, mais especificamente o nome, e o tamanho final esperado.

```
long int verifyControlData(unsigned char * data ,char * file_name, int value)
```

A primeira função é responsável por inicializar um pacote de controlo. Este vai receber um *array pack*, que vai representar o pacote, o *size* indica o nome do ficheiro, *filename* o nome deste, e o valor representa o campo de controlo do pacote. Este indica se a trama a receber é uma trama de controlo inicial, final ou se é uma trama de informação. Esta função é executada duas vezes na função loadFile(), para indicar ao receptor o início e fim da transmissão.

A segunda função é responsável por determinar se o pacote *data* é um pacote de controlo, verificando se o 1º byte tem o mesmo valor que a variável *value*. Em caso positivo, a variável *file_name* vai conter o nome do ficheiro, e a função retorna o tamanho esperado do ficheiro. Esta função é executada no início do receiveFile, para dar início à recepção do ficheiro, e é depois continuamente executada cada vez que recebe uma trama, para ver se o campo de controlo do pacote tem valor igual ao valor que indica fim.

Pacotes de informação

Os pacotes de dados/informação são pacotes cujo objetivo é de passar fragmentos de um ficheiro de um transmissor para um receptor. Tal como nos pacotes anteriores, também existem 2 funções responsáveis por isso:

```
void createDataPackage(unsigned char * pack, unsigned char * info_trama, int n_trama, long int size_trama)

int writeToFile(unsigned char * trama, int file_d, int * n_trama)
```

O createDataPackage vai escrever o conteúdo do pacote em *pack*. Este conteúdo trata-se de uma sequência de 4 bytes, seguida de uma porção de conteúdo do ficheiro, que está presente na *info_trama.* O 1º desses 4 bytes é o campo de controlo a indicar que a trama trata-se de uma trama de dados. O 2º byte é o módulo de 255 do *n_trama* (nº da trama), e os outros 2 bytes representam o tramanho da informação relativa ao ficheiro, que se segue na trama.

Ao receber do outro lado, o recetor vai pegar na trama, e envia para o writeToFile. Esta função é responsável por verificar se o campo de controlo é o de conteúdo, se o nº da trama é

igual ao *n_trama*, e de seguida, vai ler o conteúdo do ficheiro, de acordo com o tamanho da trama que está presente no pacote. Esta função vai depois escrever no ficheiro, que é representado pelo descritor *file_d.*

Envio e Recepção

As funções responsáveis por receber e enviar o ficheiro são as seguintes:

```
void receiveFile()
```

```
void loadFile(char * filename)
```

A função loadFile é responsável por ler o ficheiro com o nome igual ao *filename*, e copiar o conteúdo deste para um *array.* De seguida, envia um pacote de controlo, com o campo de controlo com o valor de inicialização, nome e tamanho do ficheiro. De seguida, vai entrar num ciclo de envio de pacotes de informação, onde vai ao *array* total do conteúdo, e vai buscar uma porção do ficheiro, sendo este tamanho escolhido inicialmente pelo utilizador, e envia através da função ll_write, presente no protocolo de ligação. À medida que se faz o ciclo, vai ser contabilizado o nº de tramas enviadas, e o tamanho enviado. Quando todo o conteúdo do ficheiro for enviado, este acaba o ciclo, e envia uma trama final, sendo esta uma trama de controlo, com conteúdo igual à inicial, com excepção do campo de controlo, que é igual ao de finalização.

A função receiveFile vai ficar em espera intermitente, até receber uma trama. Este vai verificar se se trata de um pacote de controlo, com o campo de controlo com o valor de inicialização. Em caso afirmativo, vai criar um ficheiro com o nome que recebeu da trama, e abre-o para escrita.De seguida entranum ciclo de leitura. A cada instante, este vai executar o ll_read para obter resposta do transmissor. Quando recebe, vai ver se o pacote que recebeu é um pacote de controlo, com o campo de controlo de finalização. Se não for, vai ver se se trata de uma trama de informação, invocando a função writeToFile. Em caso afirmativo, este vai filtrar o conteúdo da trama, que diz respeito ao conteúdo do ficheiro, e vai escrever no ficheiro. Caso não seja, ignora e continua neste ciclo, até ler a trama de terminação. Quando a encontrar, acaba assim a escrita do ficheiro, e compara o tamanho recebido no pacote de controlo, com o valor do ficheiro recebido.

7. Validação

Durante a realização do trabalho procuramos testa-lo de diversas maneiras. Para além de transferirmos a imagem fornecida pelo docente(pinguim.gif), testámos também utilizando outras imagens incluindo uma de maior tamanho e definição. Além das transferências normais, testámos, como viria a ser analisado mais tarde, interromper a ligação durante a transferência e introduzir resíduos raspando um fio no circuito que ligava os dois cabos de série.

8. Elementos de Valorização

Seleção de parâmetros pelo utilizador

Ao executar o programa, o apresentada uma interface ao utilizador, onde este é capaz de escolher se é o emissor ou o recetor, o *BaudRate*, qual a porta de série que vai usar, o tamanho máximo do campo de informação das tramas, o intervalo de *TimeOut* em segundos, o número máximo de retransmissões, e o nome do ficheiro a enviar.

Geração aleatória de erros em tramas de informação

Um valor ERROR_SIMULATION é definido em utils.h, que, quando tem valor igual a 1, indica ao programa para gerar erros na trama recebida, adicionando um byte com o valor de UA a uma trama de informação em cada 1000, numa posição aleatória.

Implementação de REJ

Caso a mensagem recebida tenha um erro no campo BCC2 na função llread, a trama REJ é enviada para que o emissor reenvie a mensagem que não chegou ao recetor corretamente. Esta implementação está na função int reject(unsigned char* rej) do ficheiro link.c.

Verificação da integridade dos dados pela aplicação

Após o envio é verificado o tamanho do ficheiro recebido. Esta verificação está implementada em long int verifyControlData(unsigned char * data ,char * file_name, int value) no ficheiro application.c.

Registo de ocorrências

Na estrutura de dados *Stats* declarada em utils.h são guardadas, ao longo da execução do programa, diversos valores, nomeadamente o número de tramas I enviadas e recebidas, o número de ocorrências de *TimeOuts*, e o número de REJ e RR enviados e recebidos.

9. Conclusões

Os objetivos do projeto foram cumpridos na totalidade, tendo sido implementado um protocolo de ligação de dados e um protocolo de aplicação capazes de enviar ficheiros de um computador para outro através da porta de série.

Ao longo da realização do trabalho prático foram sentidas algumas dificuldades nomeadamente com o controlo de erros e com a sincronização na execução da aplicação nos dois computadores, mas estas foram ultrapassadas.

Em suma, através da realização deste projeto conseguimos consolidar de melhor maneira alguns dos conceitos abordados nas aulas teóricas e aprofundamos o nosso conhecimento do funcionamento das comunicações em rede, através do uso da porta de série.