Ligação de Dados

Relatório do 1º trabalho laboratorial

Mestrado Integrado em Engenharia Informática e Computação

Rede de Computadores

Grupo 1

Daniel Garrido – up201403060

Nuno Castro – up201406990

Sara Santos – up201402814

18 de novembro de 2016

Conteúdo

1. Sumário

Ao longo deste relatório iremos abordar o desenvolvimento do primeiro trabalho laboratorial, Ligação de Dados, no âmbito da disciplina de Redes de Computadores. Este projeto teve como objetivo implementar um protocolo de ligação de dados de acordo com a especificação pedida e testá-lo com uma aplicação de transferência de ficheiros.

2. Introdução

O primeiro trabalho laboratorial da disciplina de Redes de Computadores consiste em implementar a ligação de dados entre dois computadores através de uma porta de série. Foi utilizado como ambiente de desenvolvimento o sistema operativo LINUX, linguagem de programação C e portas de série RS-232 de comunicação assíncrona.

De forma a garantir um serviço de comunicação de dados fiável entre os dois sistemas (emissor e recetor) foi implementado diversos protocolos que asseguram a correta transmissão dos dados tais como, sincronismo de tramas com dados organizados em *frames* limitadas com *flags*, estabelecimento e terminação da ligação, numeração de sequência de tramas e controlo de erros e fluxo, através de temporizadores e envio de tramas de confirmação (positivas ou negativas).

De seguida abordam-se de forma mais profunda a forma como abordamos este problema e como implementamos a sua resolução.

3. Arquitetura

Este projeto contém duas camadas funcionais: camada de aplicação (***Application Layer***) e camada de ligação lógica (***Link Layer***).

A ***Application Layer*** é a camada responsável pela aplicação, contendo os ficheiros *ApplicationLayer.c* e o respetivo ficheiro *header*. É dependente da camada de ligação lógica, na medida em que utilizará algumas funções desta camada. É a partir da ***Application Layer*** que será feito o envio e receção de pacotes de dados e controlo e a leitura e escrita do ficheiro a enviar.

A ***Link Layer*** é a camada responsável pela ligação entre as duas camadas, contendo os ficheiros *dataLayer.c* e o respetivo ficheiro *header*. É a partir desta camada que será feito o envio de comandos, *stuffing* e *destuffing* das mensagens, assim como o seu envio e receção.

Possui também uma interface com o utilizador por consola bastante simples, sendo que o utilizador apenas escolhe qual das máquinas é o transmissor ou recetor e indica o caminho para o ficheiro que pretender enviar. Durante a execução são apresentadas várias mensagens representativas do envio de pacotes de controlo e de comandos, bem como o número do pacote que está a ser transmitido de momento.

4. Estrutura do código

Application Layer

A camada da aplicação foi implementada nos ficheiros *ApplicationLayer.c* e *ApplicationLayer.h*. Esta camada usa uma *struct applicationLayer* (no ficheiro *utilities.h*), onde guarda o *file descriptor* do ficheiro a transmitir, o tipo da ligação, o número de tentativas até interromper a ligação e o valor do temporizador.

```c
struct applsicationLayer {
    int fileDescriptor; /*Descritor correspondente à porta série*/
    int status; /*TRANSMITTER | RECEIVER*/
    unsigned int numTransmissions;
    unsigned int timeout; /*Valor do temporizador: 1 s*/
};
```

As principais funções da camada da aplicação são:

```c
int initApplicationLayer(unsigned int timeout,unsigned int numTransmissions, char * port, int status);
int writeApp(char* filepath, unsigned int namelength);
int readApp();
int closeAppError();
int closeApplicationLayer();
```

Data Link Layer

A camada de ligação de dados foi implementada nos ficheiros *dataLayer.c* e *dataLayer.h*. Esta camada é representada pela *struct linkLayer*. Nesta está guardado o nome da porta de série utilizada, o *baudrate*, o número de tentativas até interromper a ligação, o valor do *timeout*, a trama a ser enviada, o estado atual da máquina de estados, e o número de sequência.

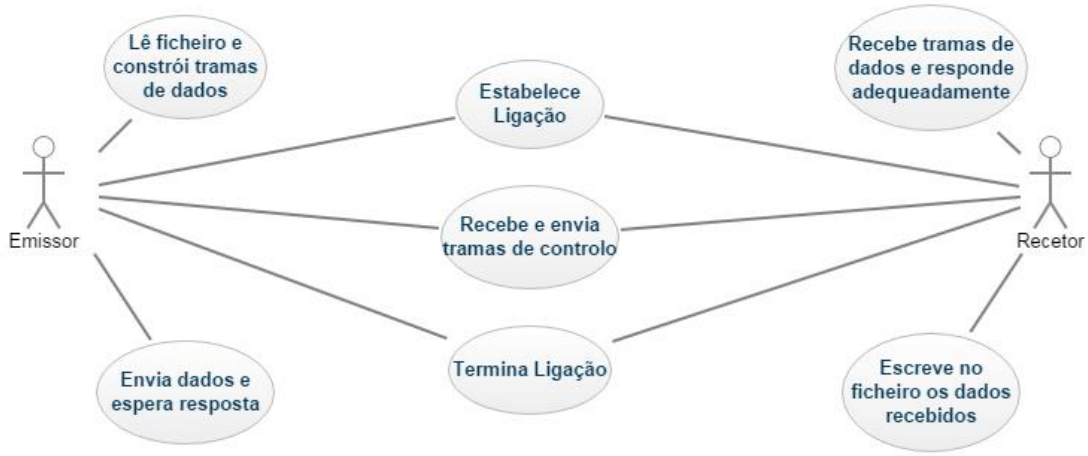
```c
struct linkLayer {
    char port[20]; /*Dispositivo /dev/ttySx, x = 0, 1*/
    int baudRate; /*Velocidade de transmissão*/
    unsigned int numTransmissions; /*Número de tentativas em caso de falha*/
    unsigned int timeout;
    char frame[MAX_SIZE]; /*Trama*/
    unsigned char machineState;
    unsigned char ns;
};
```

Estas são as funções utilizadas na *Link Layer*:

```c
void inicializa_layer(unsigned int maxTransm, unsigned int timeout);
void byte_stuffing(unsigned char *data, unsigned int i,unsigned char byte);
unsigned int byte_destuffing(unsigned char* destuff,unsigned char* stuff, unsigned int length);
int extractMessage(unsigned char* message,unsigned char* package, unsigned int length, unsigned char nr);
int llopen(char* porta,int status,int perm, unsigned int maxTransm, unsigned int timeout);
int llclose(int fd);
int llwrite(int fd, unsigned char* buffer,unsigned int length);
int llread(int fd, unsigned char* message);
int sendControl(int fd, enum Control control);
int receivePackage(int fd,unsigned char *buf);
unsigned char receiveControl(int fd,int status);
void updateStateReceiveRR(unsigned char state);
void updateStateReceiveUA(unsigned char state);
void updateStateReceiveSET(unsigned char state);
void updateStateReceiveDISC(unsigned char state);
int buildData(int fd, unsigned char *package,unsigned char *data, unsigned int size, char control);
unsigned char getBCC2(unsigned char *buffer, unsigned int length);
```

5. Casos de uso principais

Os casos de uso baseiam-se principalmente nas funcionalidades da criação e envio de tramas (de dados e controlo) de forma a enviar corretamente o ficheiro pretendido, tal como é demonstrado no diagrama seguinte.



6. Protocolo de ligação lógica

Este protocolo encontra-se implementado nos ficheiros ***dataLayer.c*** e ***dataLayer.h***. As suas principais funcionalidades são:

- Configurar a porta de série escolhida pelo utilizador;
- Estabelecer a ligação com a camada de aplicação;
- Enviar comandos e mensagens assim como receber mensagens;
- Efetuar *byte stuffing* e *byte destuffing* dos pacotes recebidos.

As principais funções desta camada são o ***llopen***, ***llclose***, ***llwrite*** e ***llread***.

A função ***llopen*** é responsável por abrir e configurar a porta de série, retornando o seu descritor. A função ***llclose*** repõe os valores originais da configuração da porta de série e fecha a ligação. A função ***llwrite*** constrói o pacote de dados, envia-o e aguarda uma resposta implementando um temporizador. A função ***llread*** lê os pacotes recebidos, validando e extraindo a mensagem, enviando uma resposta RR caso seja um pacote de dados válidos ou REJ caso contrário.

7. Protocolo de aplicação

O protocolo de aplicação está implementado nos ficheiros applicationLayer.*, ficheiros que representam a camada da aplicação, a de mais alto nível.

Esta camada está responsável por:

- Inicializar a *Application Layer*.
- Estabelecer a ligação.
- Ler o ficheiro e dividi-lo em partes e envia-lo.
- Receber e escrever a informação recebida para o ficheiro.
- Terminar a ligação.

O projeto inclui dois executáveis, um para ser utilizado pelo transmissor (*transmitter*) e outro pelo recetor (*receiver*). Desta maneira as diferentes tarefas que cada lado da ligação tem de executar ficam separadas.

O executável do transmissor executa a seguinte ordem de trabalhos:

```
30      initApplicationLayer(3, 3, argv[1],TRANSMITTER);
31      writeApp(argv[2], strlen(argv[2]));
32      closeApplicationLayer();
```

A função *initApplicationLayer* começa por inicializar os valores da *struct applicationLayer* e executa a função llopen. De seguida emite a trama *SET* para estabelecer a ligação entre as máquinas e espera pela resposta para saber que a ligação foi bem efetuada.

Em segundo lugar, a função *writeApp* abre e lê o ficheiro a transmitir para um buffer e envia a trama de controlo *START*. Assim, com o ficheiro no buffer, consegue dividi-lo em pacotes para serem enviados pela função *llwrite* e no fim envia a trama de controlo *END*.

Finalmente, a função *closeApplicationLayer* envia as tramas de controlo *DISC*, e usa a função *llclose* para encerrar a ligação da porta série.

O executável do recetor segue uma ordem de trabalhos semelhante à do transmissor:

```
21      initApplicationLayer(3, 3, argv[1], RECEIVER);
22      int res = readApp();
23      if(res != 0){
24          closeAppError();
25      }else{
26          closeApplicationLayer();
27      }
28      return 0;
```

A função *initApplicationLayer* tem uma funcionalidade semelhante à encontrada no transmissor, mas em vez de enviar a trama *SET*, aguarda que a receba e envia uma trama *UA* em resposta.

Depois, a função readApp ao receber uma trama START, cria um novo ficheiro com o nome recebido nessa trama, podendo assim receber os pacotes de informação que vai colocar no ficheiro criado. Quando recebe a trama *END*, para de ler dados da porta série.

Se *readApp* retornar um valor diferente de zero, significa que a ligação foi interrompida por tempo excessivo antes de conseguir enviar o ficheiro todo. Nesse caso a aplicação usa a função closeAppError para terminar ligação. No caso de readApp retornar zero, a função closeApplicationLayer é usada para terminar a ligação normalmente.

8. Validação

De forma a testar o funcionamento da aplicação desenvolvida, efetuou-se a transferência de alguns ficheiros: “pinguim.gif”, “pinguim.jpg” e “test.txt”. Ao longo da execução do programa vão sendo apresentadas mensagens ao utilizador acerca do pacote a enviar, isto é, no caso do recetor, se recebeu corretamente os dados (Cabeçalho, BCC2 correto ou dados repetidos) e enviou resposta (RR ou REJ), no caso do emissor se conseguiu enviar a trama, recebendo uma resposta ou se o alarme disparou.

A seguir apresentam-se duas imagens exemplificativas do funcionamento do programa no caso do transmissor e recetor, respetivamente.

```
root@ubuntu:/home/nunomiguel/Desktop/FEUP--RCOM-master/trabalho1/src# ./transmit
ter /dev/ttyS1 test.txt
Enviou set
A receber trama...
Recebeu UA
Sending START
Recebeu RR
Enviou pacote de controlo
Start sending file...
A enviar pacote nr 1
Recebeu RR
Enviou pacote nr 1
A enviar pacote nr 2
Recebeu RR
Enviou pacote nr 2
A enviar pacote nr 3
```

```
root@debian:~/Desktop/src# ./receiver /dev/ttyS1
New termios structure set
Waiting for SET...
Recebeu SET. A enviar UA...
enviou RR
START packet received
A receber pacote nr 1
enviou RR
Pacote nr 1 recebido
A receber pacote nr 2
enviou RR
Pacote nr 2 recebido
A receber pacote nr 3
enviou RR
Pacote nr 3 recebido
A receber pacote nr 4
enviou RR
Pacote nr 4 recebido
A receber pacote nr 5
enviou RR
Pacote nr 5 recebido
A receber pacote nr 6
```

9. Elementos de valorização

Implementação de REJ

A resposta REJ é enviada na função ***sendControl*** caso a função ***extractMessage*** receba um pacote que contenha o BCC2 ou cabeçalho errado. Caso isso aconteça, a camada lógica envia

```
256                 m = extractMessage(message, destuffedBuf, destuffedSize, nr);
257
258                 if(m == 0){
259                         flag_recebeu = 1;
260                         printf("enviou RR\n");
261                         sendControl(fd, RR_CONTROL);
262                         nr = (nr + 1) % 2;
263                 }else if(m == 1){
264                         printf("enviou RR rep\n");
265                         sendControl(fd, REJ_CONTROL);
266                         return -1;
267                 }else if(m == -1){
268                         sendControl(fd, REJ_CONTROL);
269                         return -1;
270                 }
```

uma mensagem de erro para a camada de aplicação avisando que o pacote que recebeu se encontra errado.

10.Conclusões

O primeiro passo para a realização do projeto foi leitura e interpretação do guião fornecido. Embora o processo descrito fosse bastante simples, a sua organização abrandou análise do mesmo. Depois de termos em mente a estrutura do programa, a fase de escreve-lo provou-se desafiadora, visto que existem vários passos desde a leitura do ficheiro até à receção do ficheiro no segundo computador.

Contudo, conseguimos concluir o objetivo principal do projeto com sucesso. Este é capaz de transferir vários tipos de ficheiros e como implementa deteção de erros é bastante robusto. No final, o projeto permitiu a consolidação dos conhecimentos adquiridos nas aulas teóricas e práticas, particularmente a separação das camadas e as suas dependências.

ANEXOS

A. ApplicationLayer.h

```
#ifndef APPLICATIONLAYER_H
#define APPLICATIONLAYER_H

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "dataLayer.h"

int initApplicationLayer(unsigned int timeout,unsigned int numTransmissions, char * port, int status);
int writeApp(char* filepath, unsigned int namelength);
int readApp();
int closeAppError();
int closeApplicationLayer();
void createInfoPackage(unsigned char *package,unsigned char *buf, int size);
int sendControlPacket(unsigned char controlFlag);
int receiveControlPacket(unsigned char controlFlag);

#endif
```

B. ApplicationLayer.c

```
#include "ApplicationLayer.h"

struct File file;
struct applicationLayer app;

int initApplicationLayer(unsigned int timeout,unsigned int numTransmissions, char * port, int status){
		app.numTransmissions = numTransmissions;
		app.timeout = timeout;
		app.status = status;
  int perm;
		if(status == TRANSMITTER)
			perm = O_RDWR | O_NOCTTY | O_NONBLOCK;
		else if( status == RECEIVER)
			perm = O_RDWR | O_NOCTTY;

  int fd = llopen(port, status,perm,numTransmissions, timeout);
  if(fd == -1){
    perror("Erro a abrir port.\n");
    exit(-1);
  }
		app.fileDescriptor = fd;

  if(status == TRANSMITTER){
    int flag_recebeu = 0;
```

```
  if(sendControl(app.fileDescriptor, SET_CONTROL) != 0) printf("Erro enviar\n");
            else printf("Enviou set\n");
            setAlarm(numTransmissions,timeout);

            printf("A receber trama...\n");
  int nTries = getNumTries();
            while(nTries < 3 && flag_recebeu == 0){
                        int state = receiveControl(app.fileDescriptor,UA_STATUS);
                        if(state == STP){
                                    flag_recebeu = 1;
                                    printf("Recebeu UA\n");
                        }else if(getAlarmeDispara()){
      if(sendControl(app.fileDescriptor, SET_CONTROL) != 0) printf("Erro enviar\n");
      setAlarmeDispara(0);
    }
     nTries = getNumTries();
            }

            if(nTries == 3)
                        printf("Transmissao falhada. A terminar...\n");

            closeAlarm();
 }else if(status == RECEIVER){
  int flag_recebeu = 0;
            printf("New termios structure set\n");

            printf("Waiting for SET...\n");

            while(!flag_recebeu){
                        int state = receiveControl(fd,SET_STATUS);
                        if(state == STP){
                                    printf("Recebeu SET. A enviar UA...\n");
                                    if(sendControl(fd,UA_CONTROL) == -1){
                                                printf("Erro no envio \n");
                                    }
                                    flag_recebeu = 1;
                        }else{
                                    printf("Não recebeu trama correta\n");
                        }
            }
 }

 return 0;
}

int writeApp(char* filepath,unsigned int namelength){
 file.fileName = filepath;
 file.fileNameLength = namelength;

            FILE* f;

            f = fopen(file.fileName, "rb");
            if(f == NULL){
                        printf("Erro na abertura do ficheiro\n");
                        return -1;
            }
            //Guarda o tamanho do ficheiro
 fseek(f, 0, SEEK_END); //Coloca apontador no final do ficheiro
 file.filelength = ftell(f); //Guarda o tamanho do ficheiro
            fseek(f,0,SEEK_SET); //Coloca apontador no inicio do ficheiro
```

```
        unsigned char *fileBuf = (unsigned char *)malloc(file.filelength);
        fread(fileBuf, sizeof(char),file.filelength, f);
        if(fclose(f)!= 0){
                printf("Erro a fechar o ficheiro\n");
                return -1;
        }

        int res = sendControlPacket(START_PACKET);
        if( res == -1){
                printf("Erro no envio do pacote START\n");
                exit(-1);
        }

        printf("Start sending file...\n");

        int maxDataSize = MAX_SIZE - 5;
        int totalPacks = ceil((float) file.filelength / maxDataSize); //tamanho de ficheiro / (256-flags)
        int bytesRem = file.filelength;
        int bytesWritten = 0;
        int i;
        unsigned char package[MAX_SIZE];
        int flag_error = 0;

        for(i = 0; i < totalPacks; i++){
                printf("A enviar pacote nr %d\n", i+1);
                int length;

                if(bytesRem < maxDataSize){
                        length = bytesRem;
                }else{
                        length = maxDataSize;
                }

                createInfoPackage(package, &fileBuf[bytesWritten], length);

                res = llwrite(app.fileDescriptor, package, length + 5);
                if(res == -1){
                        printf("Erro a enviar pacote.\n");
                        flag_error = 1;
                        break;
                }

                bytesRem -= length;
                bytesWritten += length;
                printf("Enviou pacote nr %d\n", i+1);
        }

        //Envia control END
        if(!flag_error){
                int end = sendControlPacket(END_PACKET);
                if(end == -1){
                        printf("Erro a enviar controlo END\n");
                        return -1;
                }
        }

  return 0;
}
```

```
void createInfoPackage(unsigned char *package, unsigned char *buf, int size){
	static unsigned char seqN = 0;
	unsigned char c = DATA_PACKET;
	unsigned char l2 = (size >> 8) & 0xFF;
	unsigned char l1 = size & 0xFF;

	package[0]=c;
	package[1]=seqN++;
	package[2]=l2;
	package[3]=l1;
	memcpy(&package[4],buf,size);
}

int readApp(){
   //Receive Packet START
   int start = receiveControlPacket(START_PACKET);

			if(start != 0){
				return -1;
			}

		//	printf("file length: %d\n filename: %s\n", file.filelength, file.fileName);
			//creates and opens file
			FILE *f = fopen(file.fileName, "wb");
			if(f == NULL){
				printf("Erro a criar ficheiro\n");
				return -1;
			}

			int maxDataSize = MAX_SIZE - 5;
			int totalPacks = ceil((float) file.filelength / maxDataSize); //tamanho de ficheiro /
(256-flags)
			int i = 0;
			unsigned char fileBuf[MAX_SIZE];

			for(; i < totalPacks; i++){
				printf("A receber pacote nr %d\n", i+1);

				int length = llread(app.fileDescriptor, fileBuf);
				if(length == -1){
					i--;
				}else if(length == -2){
					return -2;
				}else{
					int j = 0;
					for(; j < length - 5; j++){
						fprintf(f, "%c", fileBuf[4 + j]);
					}
					printf("Pacote nr %d recebido\n", i+1);
				}
			}

			//Acabou de ler o ficheiro
			int close = fclose(f);
			if(close != 0){
				printf("Erro a fechar o ficheiro\n");
			}

			printf("File sucessfully received\n");
```

```
                        //Espera por controlo END
                        int end = receiveControlPacket(END_PACKET);
                        if(end != 0){
                                return -1;
                        }

                        return 0;
}

int sendControlPacket(unsigned char controlFlag){
        if(controlFlag ==  START_PACKET){
                printf("Sending START\n");
        }else if(controlFlag == END_PACKET){
                printf("Sending END\n");
        }

        //Creates a new Control packet
        char aux[16];
        sprintf(aux, "%d", file.filelength);
        unsigned int size = 5 + strlen(aux) + file.fileNameLength;
        unsigned char package[size];

        package[0] = controlFlag;
  package[1] = T_SIZE; // 0 - tamanho do ficheiro
  package[2] = strlen(aux);
        unsigned i =0, j = 3;
  for(; i < strlen(aux); i++,j++){
    package[j] = aux[i];
  }
  package[j] = T_FILENAME; // 1- nome do ficheiro
  j += 1;
  package[j] = file.fileNameLength;

  j +=1;
  for(i=0; i< file.fileNameLength; i++, j++){
        package[j] = file.fileName[i];
  }

        int res = llwrite(app.fileDescriptor, package, size);
        if(res == -1){
                printf("Erro no envio do pacote de controlo\n");
                return -1;
        }else{
                printf("Enviou pacote de controlo\n");
        }
        return 0;
}

int receiveControlPacket(unsigned char controlFlag){
  unsigned char *packet = malloc(MAX_SIZE*2);
        unsigned int res = llread(app.fileDescriptor, packet);

        if(res == -1){
                printf("Erro a ler package\n");
                return -1;
        }

        if(packet[0] == START_PACKET){
                printf("START packet received\n");
        }else if(packet[0] == END_PACKET){
```

```
                printf("END packet received\n");
        }

        int l;
        if(packet[1] == T_SIZE){
                //Aloca o numero de octetos necessários para guardar length
                l = packet[2];
                char *file_length = malloc(l);
                memcpy(file_length, &packet[3], l);
                file.filelength = atoi(file_length);
                free(file_length);
        }
        int newpos = 3+l;
        if(packet[newpos] == T_FILENAME){
                newpos++;
                l = packet[newpos];

                newpos++;
                char name[l];
                int p;
                for(p = 0; p < l; p++){
                        name[p] = packet[newpos + p];
                }
                file.fileName = malloc(sizeof(char) * l);
                snprintf(file.fileName, l+1, "%s", name);
        }

        free(packet);
        return 0;
}

int closeAppError(){
          if(sendControl(app.fileDescriptor, DISC_CONTROL) == -1){ printf("Erro no envio \n");}
                return 0;
}

int closeApplicationLayer(){
  if(app.status == TRANSMITTER){
    int flag_recebeu = 0;
    if(sendControl(app.fileDescriptor, DISC_CONTROL) != 0){
                                printf("Erro a enviar DISC\n");
                                return -1;
    }else printf("Enviou DISC\n");
    setAlarm(app.numTransmissions,app.timeout);

    printf("A aguardar DISC...\n");

    int nTries = getNumTries();
    while(nTries < 3 && flag_recebeu == 0){
      int state = receiveControl(app.fileDescriptor,DISC_STATUS);
      if(state == STP){
        flag_recebeu = 1;
        printf("Recebeu DISC...A terminar\n");
        if(sendControl(app.fileDescriptor, UA_CONTROL) != 0) printf("Erro enviar\n");
      }else if(getAlarmeDispara()){
        if(sendControl(app.fileDescriptor, DISC_CONTROL) != 0) printf("Erro enviar\n");
        setAlarmeDispara(0);
      }
      nTries = getNumTries();
    }
```

```
    if(nTries == 3)
      printf("Disconexão falhada. A terminar...\n");

    closeAlarm();
  }else if(app.status == RECEIVER){
    int flag_recebeu = 0;

    printf("Waiting for DISC...\n");

    while(!flag_recebeu){
      int state = receiveControl(app.fileDescriptor, DISC_STATUS);
      if(state == STP){
        printf("Recebeu DISC. A enviar DISC...\n");
        if(sendControl(app.fileDescriptor, DISC_CONTROL) == -1){ printf("Erro no envio \n");}
        flag_recebeu = 1;
      }
    }

    setAlarm(app.numTransmissions,app.timeout);
    flag_recebeu = 0;
    int nTries = getNumTries();
    while(nTries < 3 && !flag_recebeu){
      int state = receiveControl(app.fileDescriptor, UA_STATUS);
      if(state == STP){
        printf("Recebeu UA. A terminar...\n");
        flag_recebeu = 1;
      }else if(getAlarmeDispara()){
        if(sendControl(app.fileDescriptor, DISC_CONTROL) == -1){ printf("Erro no envio \n");}
        setAlarmeDispara(0);
      }
    }
    closeAlarm();
  }

  return 0;
}
```

C. Alarme.c

```
#include "alarme.h"

int alarmeDispara = 0;
int tries;
int tentativas=0;
int time;

void handler(int signal){
  if(signal != SIGALRM) return;

  alarmeDispara = 1;
  tentativas++;
  printf("%d transmissao falhada. Retransmitir\n", tentativas);

  alarm(time);
}

void setAlarm(int nTries, int timeout){
```

```
  tries = nTries;
  time = timeout;
  struct sigaction a;
  tentativas = 0;
  a.sa_handler = handler;
  sigemptyset(&a.sa_mask);
  a.sa_flags = 0;

  sigaction(SIGALRM, &a, NULL);

  alarmeDispara = 0;

  alarm(time);
}

void closeAlarm(){
  struct sigaction a;
  a.sa_handler = NULL;
  sigemptyset(&a.sa_mask);
  a.sa_flags = 0;

  sigaction(SIGALRM, &a, NULL);

  alarm(0);
}

int getNumTries(){
  return tentativas;
}

int getAlarmeDispara(){
  return alarmeDispara;
}

void setAlarmeDispara(int value){
  alarmeDispara = value;
}
```

D. Alarme.h

```
#ifndef ALARME_H
#define ALARME_H

#include <signal.h>
#include <stdio.h>
#include <unistd.h>

extern int alarmeDispara;

void handler(int signal);
void setAlarm(int nTries, int timeout);
void closeAlarm();
int getNumTries();
int getAlarmeDispara();
void setAlarmeDispara(int value);

#endif
```

E. dataLayer.c

```
#include "dataLayer.h"

struct linkLayer layer;
struct termios oldtio,newtio;

volatile int STOP=FALSE;

void inicializa_layer(unsigned int maxTransm, unsigned int timeout){
	strcpy(layer.port, MODEMDEVICE);
	layer.baudRate = BAUDRATE;
	layer.machineState = START;
	layer.ns = 0;
	layer.numTransmissions = maxTransm;
	layer.timeout = timeout;
}

void byte_stuffing(unsigned char *data, unsigned int i,unsigned char byte){
	data[i] = ESCAPE;
	data[i++] = byte ^ 0x20;
}

unsigned int byte_destuffing(unsigned char* destuff,unsigned char* stuff, unsigned int length){
	int i = 0, j = 0;

	for(i = 0; i < length; i++, j++){
		if(stuff[i] == ESCAPE){
			i++;
			destuff[j] = (stuff[i] ^ 0x20);
		}else{
			destuff[j] = stuff[i];
		}
	}

	return j;

}

int extractMessage(unsigned char* message,unsigned char* package, unsigned int length, unsigned char
nr){
	unsigned char tempBCC2 = 0x00;
	unsigned char ns;
	int i;

	ns = (nr + 1) % 2;
	if((package[2] >> 6) == nr){
		printf("Pacote repetido.\n");
		return 1;
	}else if(package[0] != FLAG
					|| package[1] != A
					|| package[2] !=(unsigned char) (ns << 6)
					|| package[3] != (package[1]^package[2])
					|| package[length-1] != FLAG){
		printf("Cabeçalho errado.\n");
		return -1;
	}

	i = 4;
```

```
		for(; i < length-2; i++){
			message[i - 4] = package[i];
			tempBCC2 = tempBCC2 ^ (unsigned char)package[i];
		}

		if(tempBCC2 != package[length-2]){
			printf("BCC2 errado\n");
			return -1;
		}

		return 0;
}

int llopen(char* porta,int status, int perm, unsigned int maxTransm, unsigned int timeout){
		inicializa_layer(maxTransm, timeout);
		int fd;
		if(status == TRANSMITTER){
			fd = open(porta,perm);
			if (fd <0) {perror("porta"); exit(-1);}
		}else if(status == RECEIVER){
			fd = open(porta, perm);
			if (fd <0) {perror("porta"); exit(-1);}
		}

		if ( tcgetattr(fd,&oldtio) == -1) { /* save current port settings */
		  perror("tcgetattr");
		  exit(-1);
		}

		bzero(&newtio, sizeof(newtio));
		newtio.c_cflag = layer.baudRate | CS8 | CLOCAL | CREAD;
		newtio.c_iflag = IGNPAR;
		newtio.c_oflag = 0;

		/* set input mode (non-canonical, no echo,...) */
		newtio.c_lflag = 0;
		newtio.c_cc[VTIME]    = 0;   /* inter-character timer unused */
		newtio.c_cc[VMIN]     = 1;   /* blocking read until 1 chars received */
 /*
		VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a
		leitura do(s) pr�ximo(s) caracter(es)
 */

		tcflush(fd, TCIOFLUSH);

		if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
		  perror("tcsetattr");
		  exit(-1);
		}
		return fd;
}

int llclose(int fd){
		tcsetattr(fd,TCSANOW,&oldtio);
		return close(fd);
}

int buildData(int fd,unsigned char *package,unsigned char *data, unsigned int size, char control){
		//Creates data package
		unsigned char bcc1 = A^control;
```

```
        unsigned char bcc2 = getBCC2(data, size);

        int i = 0, j;
        package[i] = FLAG;
        i++;
        package[i] = A;
        i++;
        package[i] = control;
        i++;
        if(bcc1 == FLAG || bcc1 == ESCAPE){
                package[i] = ESCAPE;
                i++;
                package[i] = bcc1 ^ 0x20;
                i++;
        }else{
                package[i] = bcc1;
                i++;
        }

        for(j = 0; j < size; j++,i++){
                if(data[j] == FLAG || data[j] == ESCAPE){
                        package[i] = ESCAPE;
                        i++;
                        package[i] = data[j] ^ 0x20;
                }else{
                        package[i] = data[j];
                }
        }


        if(bcc2 == FLAG || bcc2 == ESCAPE){
                package[i] = ESCAPE;
                i++;
                package[i] = bcc2 ^ 0x20;
                i++;
        }else{
                package[i] = bcc2;
                i++;
        }

        package[i] = FLAG;
        i++;
        return i;
}

unsigned char getBCC2(unsigned char *buffer, unsigned int length){
        unsigned char temp = 0x00;
        int i=0;
        for(;i < length;  i++){
                temp ^= buffer[i];
        }
        return temp;
}

int isRR(char control[]){
        if(control[0] == FLAG &&
                control[1] == A &&
                control[2] == C_RR &&
                control[3] == (A ^ C_RR) &&
                control[4] == FLAG){
```

```
                return 1;
        }
        return 0;
}

int isREJ(char control[]){
        if(control[0] == FLAG &&
                control[1] == A &&
                control[2] == C_REJ &&
                control[3] == (A ^ C_REJ) &&
                control[4] == FLAG){
                return 1;
        }
        return 0;
}

int llwrite(int fd,unsigned char* buffer, unsigned int length){
        static unsigned char ns = 0;
        int flag_recebeu = 0;
        unsigned char package[MAX_SIZE * 2];
        int size = buildData(fd, package, buffer, length, (ns << 6));

        int res = write(fd, package, size);
        sleep(1);
        if(res == 0 || res == -1){
                printf("Erro a enviar dados\n");
                return -1;
        }
        setAlarm(layer.numTransmissions, layer.timeout);

        int nTries = getNumTries();

        while(!flag_recebeu && nTries < layer.numTransmissions){
                receiveControl(fd, RR_STATUS);

                if(layer.machineState == STP){
                        printf("Recebeu RR\n");
                        flag_recebeu = 1;
                        closeAlarm();
                }else if(getAlarmeDispara()){
                        printf("Alarme disparou\n");
                        res = write(fd, package, size);
                        sleep(1);
                        if(res == 0 || res == -1){
                                printf("Erro a enviar dados\n");
                        }
                        setAlarmeDispara(0);
                }
        //      tcflush(fd, TCIOFLUSH);
                nTries = getNumTries();
        }

        ns = (ns + 1) % 2;

        if(nTries ==  layer.numTransmissions){
                printf("Conexão Falhada. Terminar...\n");
                return -1;
        }

        return 0;
```

```
}

int llread(int fd,unsigned char *message){
	static unsigned char nr = 1;
	unsigned char buf[MAX_SIZE * 2];
	int flag_recebeu = 0;
	int size, destuffedSize, m;
	while(!flag_recebeu){
		size = receivePackage(fd, buf);
		if(size == -1){
			return size;
		}else if(size == -2){
			printf("Recebeu DISC\n");
			return -2;
		}

		unsigned char destuffedBuf[MAX_SIZE];
		destuffedSize = byte_destuffing(destuffedBuf, buf, size);

		m = extractMessage(message, destuffedBuf, destuffedSize, nr);

		if(m == 0){
			flag_recebeu = 1;
			printf("enviou RR\n");
			sendControl(fd, RR_CONTROL);
			nr = (nr + 1) % 2;
		}else if(m == 1){
			printf("enviou RR rep\n");
			sendControl(fd, REJ_CONTROL);
			return -1;
		}else if(m == -1){
			sendControl(fd, REJ_CONTROL);
			return -1;
		}
	}

	return (destuffedSize - 6); //retorna tamanho da mensagem: tamanho total - flags
}

int receivePackage(int fd,unsigned char* buf){
	enum MachineStates state = STR;
	int i = 0, res;
	unsigned char byte;

	while(state != ST){
		res = read(fd, &byte, 1);
		if(res == -1 || res == 0){
			printf("Erro na leitura do pacote\n");
			return -1;
		}

		switch (state) {
			case STR:
				if(byte == FLAG){
					buf[i] = byte;
					i++;
					state = FLAG_RCV;
				}
				break;
			case FLAG_RCV:
```

```
                                        if(byte == A){
                                                buf[i] = byte;
                                                i++;
                                                state = A_RCV;
                                        }else if(byte != FLAG){
                                                state = STR;
                                        }
                                        break;
                                case A_RCV:
                                        if(byte == C_DISC){
                                                return -2;
                                        }
                                        if(byte != FLAG){
                                                buf[i] = byte;
                                                i++;
                                                state = C_RCV;
                                        }else{
                                                state = FLAG_RCV;
                                        }
                                        break;
                                case C_RCV:
                                        if(byte != FLAG){
                                                buf[i] = byte;
                                                i++;
                                                state = BCC_OK;
                                        }else{
                                                state = FLAG_RCV;
                                        }
                                        break;
                                case BCC_OK:
                                        if(byte == FLAG){
                                                buf[i] = byte;
                                                i++;
                                                state = ST;
                                        }else{
                                                buf[i] = byte;
                                                i++;
                                        }
                                        break;
                                default:
                                        break;
                        }
                }

                return i;
}

int sendControl(int fd, enum Control control){
                unsigned char SET[] = {FLAG, A, C_SET, A^C_SET, FLAG};
                unsigned char UA[] = {FLAG, A, C_UA, A^C_UA, FLAG};
                unsigned char DISC[] = {FLAG, A, C_DISC, A^C_DISC, FLAG};
                unsigned char RR[] = {FLAG, A, C_RR, A^C_RR, FLAG};
                unsigned char REJ[] = {FLAG, A, C_REJ, A^C_REJ, FLAG};

                int res;
                switch (control) {
                        case SET_CONTROL:
                                res = write(fd, SET,CONTROL_LENGTH);
                                sleep(1);
                                if( res == 0 || res == -1){
```

```
                                        printf("Erro a enviar \n");
                                        return -1;
                                }
                                break;
                        case UA_CONTROL:
                                res = write(fd, UA,CONTROL_LENGTH);
                                sleep(1);
                                if( res == 0 || res == -1){
                                        printf("Erro a enviar \n");
                                        return -1;
                                }
                                break;
                        case DISC_CONTROL:
                                res = write(fd, DISC,CONTROL_LENGTH);
                                sleep(1);
                                if( res == 0 || res == -1){
                                        printf("Erro a enviar \n");
                                        return -1;
                                }
                                break;
                        case RR_CONTROL:
                                res = write(fd, RR,CONTROL_LENGTH);
                                sleep(1);
                                if( res == 0 || res == -1){
                                        printf("Erro a enviar \n");
                                        return -1;
                                }
                                break;
                        case REJ_CONTROL:
                                res = write(fd, REJ,CONTROL_LENGTH);
                                sleep(1);
                                if( res == 0 || res == -1){
                                        printf("Erro a enviar \n");
                                        return -1;
                                }
                                break;
                        default:
                                return -1;
                                break;
                }
                return 0;
}

unsigned char receiveControl(int fd, int status){
                STOP = FALSE;
                layer.machineState = START;
                unsigned char set[5];
                int i = 0;
                read(fd,set+i,1);

                if(status == SET_STATUS) updateStateReceiveSET(set[i]);
                else if(status == UA_STATUS) updateStateReceiveUA(set[i]);
                else if(status == DISC_STATUS) updateStateReceiveDISC(set[i]);
                else if(status == RR_STATUS) updateStateReceiveRR(set[i]);

                i++;

                if(set[i-1] == FLAG){
                        while (STOP==FALSE){
                                read(fd,set+i,1);
```

```
                                        if(status == SET_STATUS) updateStateReceiveSET(set[i]);
                                        else if(status == UA_STATUS) updateStateReceiveUA(set[i]);
                                        else if(status == DISC_STATUS) updateStateReceiveDISC(set[i]);
                                        else if(status == RR_STATUS) updateStateReceiveRR(set[i]);

                                        i++;
                                        if(set[i-1] == FLAG){
                                                STOP = TRUE;
                                        }
                                }
                        }

                        return layer.machineState;
}

void updateStateReceiveRR(unsigned char state){
                        switch(state){
                        case FLAG:
                                if(layer.machineState == START){
                                        layer.machineState = FLAG;
                                }else if(layer.machineState == (A ^ C_RR)){
                                        layer.machineState = STP;
                                }else{
                                        layer.machineState = START;
                                }
                                break;
                        case A:
                                if(layer.machineState == FLAG){
                                        layer.machineState = A;
                                }else{
                                        layer.machineState = START;
                                }
                                break;
                        case C_RR:
                                if(layer.machineState == A){
                                        layer.machineState = C_RR;
                                }else{
                                        layer.machineState = START;
                                }
                                break;
                        case (A ^ C_RR):
                                if(layer.machineState == C_RR){
                                        layer.machineState = A ^ C_RR;
                                }else{
                                        layer.machineState = START;
                                }
                                break;
                        default:
                                layer.machineState = START;
                                break;
                        }
}

void updateStateReceiveUA(unsigned char state){
                        switch(state){
                        case FLAG:
                                if(layer.machineState == START){
                                        layer.machineState = FLAG;
                                }else if(layer.machineState == (A ^ C_UA)){
                                        layer.machineState = STP;
```

```
                }else{
                        layer.machineState = START;
                }
                break;
        case A:
                if(layer.machineState == FLAG){
                        layer.machineState = A;
                }else{
                        layer.machineState = START;
                }
                break;
        case C_UA:
                if(layer.machineState == A){
                        layer.machineState = C_UA;
                }else{
                        layer.machineState = START;
                }
                break;
        case (A ^ C_UA):
                if(layer.machineState == C_UA){
                        layer.machineState = A ^ C_UA;
                }else{
                        layer.machineState = START;
                }
                break;
        default:
                layer.machineState = START;
                break;
        }
}

void updateStateReceiveSET(unsigned char state){
        switch(state){
        case FLAG:
                if(layer.machineState == START){
                        layer.machineState = FLAG;
                }else if(layer.machineState == (A ^ C_SET)){
                        layer.machineState = STP;
                }else{
                        layer.machineState = START;
                }
                break;
        case C_SET:
                if(layer.machineState == FLAG){
                        layer.machineState = A;
                }else if(layer.machineState == A){
                        layer.machineState = C_SET;
                }else{
                        layer.machineState = START;
                }
                break;
        case (A ^ C_SET):
                if(layer.machineState == C_SET){
                        layer.machineState = A ^ C_SET;
                }else{
                        layer.machineState = START;
                }
                break;
        default:
                layer.machineState = START;
```

```
                        break;
                }
}

void updateStateReceiveDISC(unsigned char state){
                switch(state){
                case FLAG:
                        if(layer.machineState == START){
                                layer.machineState = FLAG;
                        }else if(layer.machineState == (A ^ C_DISC)){
                                layer.machineState = STP;
                        }else{
                                layer.machineState = START;
                        }
                        break;
                case A:
                        if(layer.machineState == FLAG){
                                layer.machineState = A;
                        }else{
                                layer.machineState = START;
                        }
                        break;
                case C_DISC:
                        if(layer.machineState == A){
                                layer.machineState = C_DISC;
                        }else{
                                layer.machineState = START;
                        }
                        break;
                case (A ^ C_DISC):
                        if(layer.machineState == C_DISC){
                                layer.machineState = A ^ C_DISC;
                        }else{
                                layer.machineState = START;
                        }
                        break;
                default:
                        layer.machineState = START;
                        break;
                }
}
```

F. dataLayer.h

```
#ifndef DATALAYER_H
#define DATALAYER_H

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <signal.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "utilities.h"
```

```
#include "alarme.h"

struct linkLayer {
        char port[20]; /*Dispositivo /dev/ttySx, x = 0, 1*/
        int baudRate; /*Velocidade de transmissão*/
        unsigned int numTransmissions; /*Número de tentativas em caso de falha*/
        unsigned int timeout;
        char frame[MAX_SIZE]; /*Trama*/
        unsigned char machineState;
        unsigned char ns;
};

extern struct linkLayer layer;

void inicializa_layer(unsigned int maxTransm, unsigned int timeout);
void byte_stuffing(unsigned char *data, unsigned int i,unsigned char byte);
unsigned int byte_destuffing(unsigned char* destuff,unsigned char* stuff, unsigned int length);
int extractMessage(unsigned char* message,unsigned char* package, unsigned int length, unsigned char
nr);
int llopen(char* porta,int status,int perm, unsigned int maxTransm, unsigned int timeout);
int llclose(int fd);
int llwrite(int fd, unsigned char* buffer,unsigned int length);
int llread(int fd, unsigned char* message);
int sendControl(int fd, enum Control control);
int receivePackage(int fd,unsigned char *buf);
unsigned char receiveControl(int fd,int status);
void updateStateReceiveRR(unsigned char state);
void updateStateReceiveUA(unsigned char state);
void updateStateReceiveSET(unsigned char state);
void updateStateReceiveDISC(unsigned char state);
int buildData(int fd, unsigned char *package,unsigned char *data, unsigned int size, char control);
unsigned char getBCC2(unsigned char *buffer, unsigned int length);


#endif
```

G. noncanonical.c

```
/*Non-Canonical Input Processing*/

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "ApplicationLayer.h"

int main(int argc, char** argv){
        if ( (argc < 2) ||
                        ((strcmp("/dev/ttyS0", argv[1])!=0) &&
                         (strcmp("/dev/ttyS1", argv[1])!=0) )) {
          printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
          exit(1);
        }
```

```
	initApplicationLayer(3, 3, argv[1], RECEIVER);
	int res = readApp();
	if(res != 0){
		closeAppError();
	}else{
		closeApplicationLayer();
	}
	return 0;
}
```

H. utilities.h

```
#ifndef UTILITIES_H
#define UTILITIES_H

#define TRANSMITTER		0
#define RECEIVER	1
#define MAX_SIZE	256
#define BAUDRATE B9600
#define MODEMDEVICE  "/dev/ttyS1"
#define CONTROL_LENGTH 5
#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define FALSE		0
#define TRUE		1
#define FLAG		0x7E
#define A		0x03
#define A_RESPONSE		0x01
#define C_SET		0x03
#define C_UA		0x07
#define C_DISC  0x0B
#define C_RR 0x05
#define C_REJ 0x01
#define START		1
#define STP		2
#define SET_STATUS 0
#define DISC_STATUS 1
#define UA_STATUS 2
#define RR_STATUS 3
#define REJ_STATUS 4

//BYTE STUFFING
#define ESCAPE		0x7d
#define XORFLAG	(FLAG ^ 0x20)
#define XORESC		(ESCAPE ^ 0x20)

//CONTROL FLAG C
#define START_PACKET 2
#define END_PACKET		3
#define DATA_PACKET 1

//CONTROL FLAG T
#define T_SIZE		0
#define T_FILENAME		1

enum MachineStates{
	STR, FLAG_RCV, A_RCV, C_RCV, BCC_OK, ST
};
```

```
struct applicationLayer {
        int fileDescriptor; /*Descritor correspondente à porta série*/
        int status; /*TRANSMITTER | RECEIVER*/
        unsigned int numTransmissions;
        unsigned int timeout; /*Valor do temporizador: 1 s*/
};

struct File{
  unsigned int fileNameLength;
  char* fileName;
  unsigned int filelength;
  char* buf;
};

enum Control{
        SET_CONTROL,
        UA_CONTROL,
        RR_CONTROL,
        REJ_CONTROL,
        DISC_CONTROL,
        ERROR_CONTROL
};

#endif
```

I. writenoncanonical.c

```
/*Non-Canonical Input Processing*/

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "ApplicationLayer.h"

struct linkLayer layer;

int main(int argc, char** argv){
   if ( (argc < 3) ||
                ((strcmp("/dev/ttyS0", argv[1])!=0) &&
                 (strcmp("/dev/ttyS1", argv[1])!=0) )) {
     printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1 filepath\n");
     exit(1);
   }
   FILE *f;
   f = fopen(argv[2],"r");
   if(f == NULL){
     printf("Ficheiro inválido\n");
     exit(1);
   }
   fclose(f);
            initApplicationLayer(3, 3, argv[1],TRANSMITTER);
```

```
    writeApp(argv[2], strlen(argv[2]));
    closeApplicationLayer();

    return 0;
}
```