U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Redes de Computadores

1º Trabalho Laboratorial

Mestrado Integrado em Engenharia Informática e Computação

(7 de novembro de 2017)

Bárbara Silva **up201505628**@fe.up.pt

Catarina Ferreira **up201506671**@fe.up.pt

Julieta Frade **up201506530**@fe.up.pt

Índice

Sumário

Este relatório foi elaborado no âmbito da unidade curricular de Redes e Computadores, e trata-se de uma complementação ao primeiro trabalho laboratorial, cuja essência é a transferência de dados. O trabalho consiste no desenvolvimento de uma aplicação capaz de transferir ficheiros de um computador para o outro através de uma porta série.

Isto posto, o trabalho foi concluído com sucesso, visto que todos os objetivos estabelecidos foram cumpridos e foi finalizada uma aplicação perfeitamente funcional e capaz de transferir ficheiros sem perda de dados.

Introdução

O objetivo deste trabalho é implementar um protocolo de ligação de dados, de acordo com o guião fornecido, e testar o protocolo com uma aplicação simples de transferência de ficheiros, recorrendo a uma porta série. Quanto ao relatório, o seu objetivo é expor e explicar toda a componente teórica presente neste primeiro trabalho, tendo a seguinte estrutura:

- **Arquitetura**

  Exibição dos blocos funcionais e interfaces presentes.

- **Estrutura do código**

  Demonstração das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura.

- **Casos de uso principais**

  Identificação dos mesmos e demonstração das sequências de chamada de funções.

- **Protocolo de ligação lógica**

  Identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código.

- **Protocolo de aplicação**

  Identificação dos principais aspetos funcionais e descrição da estratégia de implementação dos mesmos com apresentação de extratos de código.

- **Validação**

  Descrição dos testes efetuados com apresentação quantificada dos resultados.

- **Eficiência do protocolo de ligação de dados**

  Caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.

- **Conclusão**

  Síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados.

Arquitetura

O trabalho está dividido em dois blocos funcionais, focando-se no emissor e no recetor. Cada um destes blocos incorpora a camada de ligação de dados e de aplicação isoladamente. Note-se que, foi decidido dividir desta forma com o intuito de isolar os casos.

Estrutura de código

O código está dividido em dois ficheiros de código, correspondentes às funções necessárias para execução do programa. Assim existe o ficheiro *writer.c* – responsável pelas funções do emissor - e o ficheiro *reader.c* – responsável pelas funções do recetor. Para ambos os ficheiros existe ainda um *header file* no qual estão declaradas todas as funções necessárias.

Writer

Funções principais da camada de ligação:

- **llopen** – envia trama de supervisão *SET* e recebe trama *UA*
- **llwrite** – realiza *stuffing* das tramas I e envia-as.
- **llclose** – envia trama de supervisão *DISC*, recebe *DISC* e envia *UA*.

Funções principais da camada de aplicação:

- **main** – base da camada de aplicação pois é esta que controla todo o processo que ocorre nesta camada e que faz as chamadas às funções da camada de ligação.
- **openReadFile** – abre um ficheiro e lê o seu conteúdo.
- **splitMessage** – divide uma mensagem proveniente do ficheiro em *packets*.

Variáveis globais:

- **sumAlarms** – contador de alarmes, inicializada a 0.
- **flagAlarm** – passa a *TRUE* quando é acionado o alarme, inicializada a *FALSE*.
- **trama** – número sequencial da trama (*Ns*) a enviar, inicializada a 0.
- **paragem** – passa a *TRUE* quando *UA* for recebido, inicializada a *FALSE*.
- **numMensagens** – número de sequência das tramas I (*N*), inicializada a 0.
- **numTotalTramas** – contador de tramas enviadas, inicializada a 0.
- **oldtio, newtio** – structs termios com as definições da porta série.

Macros pertinentes:

- **BAUDRATE –** Capacidade de ligação**.**
- **NUMMAX –** Número máximo de tentativas de reenvio.
- **TIMEOUT** – Número de segundos de cada alarme
- **sizePacketConst –** Número de bytes em cada *packet*.
- **bcc1ErrorPercentage –** Percentagem de erros gerados no *BCC1*.
- **bcc2ErrorPercentage –** Percentagem de erros gerados no *BCC2*.

<u>Reader</u>

Funções principais da camada de ligação:

- **llopen** – lê trama de controlo *SET* e envia a trama *UA*.
- **llread** – lê tramas I e faz *destuffing*.
- **llclose** – lê trama de controlo *DISC*, envia *DISC* de volta e recebe *UA*.

Funções principais da camada de aplicação:

- **main** – base da camada de aplicação pois é esta que controla todo o processo que ocorre nesta camada e que faz as chamadas às funções da camada de ligação.
- **createFile** – cria ficheiro com os dados recebidos nas tramas I.

Variáveis globais:

- **esperado** – trama (*Nr*) esperada, usada para o tratamento de duplicados, inicializada a 0.
- **oldtio, newtio** – structs termios com as definições da porta série.

Macros pertinentes:

- **BAUDRATE –** Capacidade de ligação**.**

Casos de uso principais

Os principais casos de uso desta aplicação são: a interface, que permite ao transmissor escolher o ficheiro a enviar, e a transferência desse mesmo ficheiro, via porta série, entre dois computadores, o transmissor e o recetor.

Quanto à interface, o utilizador, de forma a dar início à aplicação, deverá inserir um conjunto de argumentos. Sendo emissor, deverá inserir qual a porta série a ser utilizada (ex: **/dev/ttyS0)**, e o ficheiro a ser enviado (ex: **pinguim.gif**). Sendo recetor, basta inserir a porta série.

A transmissão de dados dá-se com a seguinte sequência:

- Transmissor escolhe o ficheiro a enviar.
- Configuração da ligação entre os dois computadores.
- Estabelecimento da ligação.
- Transmissor envia dados.
- Recetor recebe os dados.
- Recetor guarda os dados num ficheiro com o mesmo nome do ficheiro enviado pelo emissor.
- Terminação da ligação.

Protocolo de ligação lógica

LLOPEN

```
int LLOPEN(int fd, int x);
```

Esta função tem a responsabilidade de estabelecer a ligação entre o emissor e o recetor.

No emissor, esta função envia a trama de controlo *SET* e ativa o temporizador que é desativado depois de receber resposta (*UA*). Se não receber resposta dentro de um tempo *time-out*, *SET* é reenviado. Este mecanismo de retransmissão só é repetido um número máximo de vezes, se este número for atingido o programa termina.

No recetor, esta função espera pela chegada de uma trama de controlo SET para responder com um UA.

Para enviar tanto o *UA* como o *SET* é usada a função ***sendControlMessage***. Esta função recebe como argumento o campo de controlo das tramas de Supervisão, o que permite que a função seja usada para enviar qualquer tipo de trama deste tipo.

Como o mecanismo de receção do *UA* necessita de temporizador, usa-se a função ***stateMachineUA*** que tem o que é necessário para tal. Na leitura do *SET* é usada a função ***readControlMessage*** que recebe como argumento o campo de controlo das tramas de Supervisão para verificar se a trama recebida é de facto do tipo que é desejado.

As escritas são feitas trama a trama, no entanto a leitura é feita carater a carater.

LLWRITE

```
int LLWRITE(int fd, unsigned char *mensagem, int size);
```

Esta é a função no emissor responsável pelo envio das tramas e pelo *stuffing* das mesmas.

Primeiro é feito o *framing* da mensagem, ou seja, acrescentado o cabeçalho do Protocolo de Ligação à mensagem (para calcular o *BCC2* é chamada a função ***calculoBCC2***). Seguidamente é feito o *stuffing* da mensagem e do *BCC2* (feito na função ***stuffingBCC2***). Posto isto, a trama está pronta a ser enviada. Esta escrita é feita trama a trama.

Antes da escrita são introduzidos erros pelas funções ***messUpBCC1*** e ***messUpBCC2***. Estas funções substituem o conteúdo de uma posição aleatória (no caso do *BCC1* da posição 1 a 3 e no caso do *BCC2* nas posições ocupadas pelo campo de dados e *BCC2*) com uma letra aleatória, tendo em conta uma probabilidade erro escolhida.

O envio da trama tem o mesmo mecanismo de *time-out* e retransmissão que o envio do *SET* no ***llopen***. Ou seja, depois de enviar a trama é acionado um alarme até à receção de uma resposta (*RR* ou *REJ*) e se atingido esse alarme a mensagem é reenvida (mecanismo que se pode ocorrer um numero máximo de vezes). Se recebido um *REJ* a mensagem é reenviada. Para fazer a verificação se é *REJ* ou *RR* é usada a função ***readControlMessageC*** que retorna o campo de controlo da trama de Supervisão lida.

LLREAD

```
unsigned char *LLREAD(int fd, int *sizeMessage);
```

Esta é a função no recetor responsável pela receção das tramas e pelo *destuffing* das mesmas.

A leitura é feita carater a carater. Para verificar o *BCC2* é usada a função ***checkBCC2***, caso esteja correto é enviado *RR*, caso contrário *REJ* usando a função ***sendControlMessage***. O campo de controlo enviado depende do número de

sequência da trama (*Nr*). Seguidamente é feito o *destuffing* do campo de dados. Também é feita uma verificação se o número de sequência de tramas é o esperado para ser possível tratar duplicados.

LLCLOSE

```
void LLCLOSE(int fd);
```

Esta função tem a responsabilidade de terminar a ligação entre o emissor e o recetor.

No emissor, é enviado a trama de Supervisão *DISC* com ajuda da função ***sendControlMessage*** (que recebe como argumento o campo de controlo da trama a enviar) e esperado outro *DISC* de volta pela função ***readControlMessageC*** (que retorna o campo de controlo da trama lida). Para finalizar é enviado um *UA*.

No recetor é esperado um *DISC*, enviado um *DISC* e esperado um UA com as funções ***readControlMessage*** e ***sendControlMessage*** (que recebem ambas o campo de controlo pretendido).

Protocolo de aplicação

O protoloco de aplicação implementado tem como aspetos principais:

- O envio dos pacotes de controlo START e END. Estes contêm o nome e o tamanho do ficheiro a ser enviado;
- A divisão do ficheiro em fragmentos quando se trata do emissor e a concatenação dos fragmentos recebidos, quando se trata do recetor;
- Encapsular cada fragmento de dados com um header contendo o número de sequência do pacote (módulo 255) e o tamanho do fragmento;
- Leitura do ficheiro a enviar, quando se trata do emissor, e criação do ficheiro, quando se trata do recetor.

Estas funcionalidades foram implementadas usando funções descritas a seguir.

controlPackageI

```
unsigned char *controlPackageI(unsigned char state, off_t sizeFile,
unsigned char *fileName, int sizeOfFilename, int *sizeControlPackageI);
```

Esta função retorna um pacote START ou END. Recebendo como argumentos um caracter "state" para identificar se o pacote pretendido é START ou END, o nome do ficheiro e o tamanho do ficheiro. Este pacote será enviado usando a função LLWRITE, pertencente ao protocolo de ligação de dados.

splitMessage

```
unsigned char *splitMessage(unsigned char *mensagem, off_t *indice, int
*sizePacket, off_t sizeFile);
```

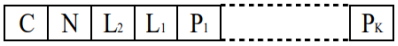
Esta função recebe como argumentos o conteúdo do ficheiro, o índice do primeiro elemento pertencente ao pacote, o tamanho do pacote e o tamanho do ficheiro.

Esta função retorna um pacote de tamanho sizePacket que poderá ser reduzido dentro desta função se não existir sizePacket elementos desde o índice até ao final do conteúdo do ficheiro.

headerAL

```
unsigned char *headerAL(unsigned char *mensagem, off_t sizeFile, int
*sizePacket);
```

Esta função retorna a mensagem a ser enviada concatenando o header com o pacote recebido. A mensagem vai então ficar da seguinte forma:



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

O header é constituído pelos caracteres: C, N, L2 e L1 e o pacote recebido são os caracteres de P1 a PK.

openReadFile

```
unsigned char *openReadFile(unsigned char *fileName, off_t *sizeFile);
```

Esta função retorna o conteúdo do ficheiro a ser enviado e, como argumento, o tamanho do ficheiro. Recebe como argumento o nome do ficheiro que é escrito aquando da chamada do programa.

createFile

```
void createFile(unsigned char *mensagem, off_t *sizeFile, unsigned char
*filename);
```

Esta função é responsável pela criação do ficheiro, usando o conteúdo recebido através dos pacotes de dados. O ficheiro criado tem o mesmo nome do ficheiro transmitido, obtido através da trama START.

Validação

De forma a estudar a aplicação desenvolvida, foram efetuados os seguintes testes:

- Envio de ficheiros de vários tamanhos.
- Geração de curto circuito enquanto se envia um ficheiro.
- Interrupção da ligação por alguns segundos enquanto se envia um ficheiro.
- Envio do um ficheiro com variação na percentagem de erros simulados.
- Envio do um ficheiro com variação do tamanho de pacotes.
- Envio do um ficheiro com variação das capacidades de ligação (*baudrate*).

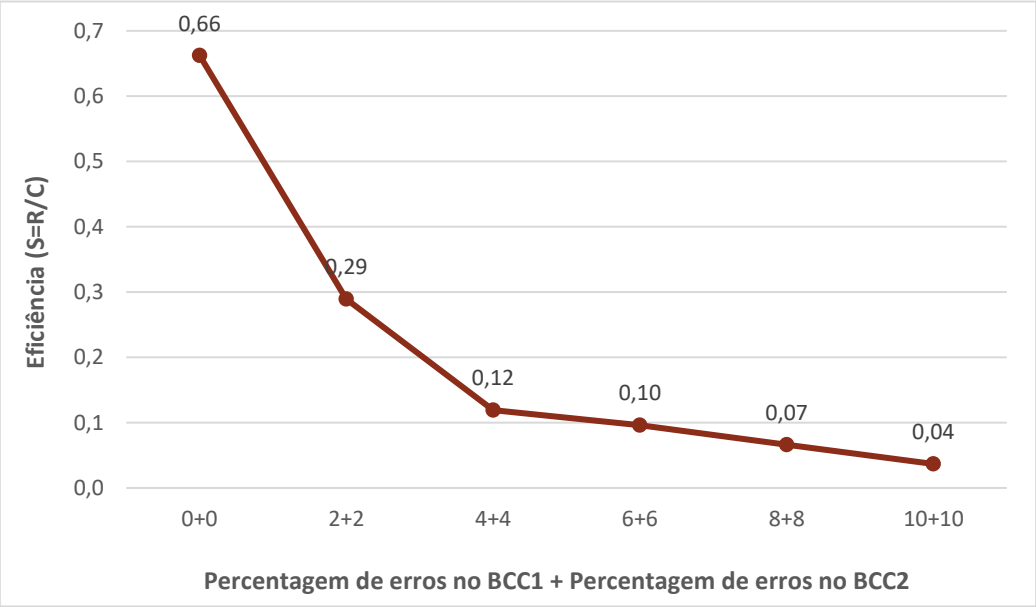
Todos os testes foram concluídos com sucesso.

Eficiência do protocolo de ligação de dados

De forma a avaliar a eficiência do protocolo desenvolvido, foram feitos os seguintes três testes e elaborado, respetivamente, uma tabela e um gráfico. Para as mesmas condições foram feitos sempre dois testes e a sua média para diminuir o desvio dos dados. Todas as tabelas estão presentes no anexo I.

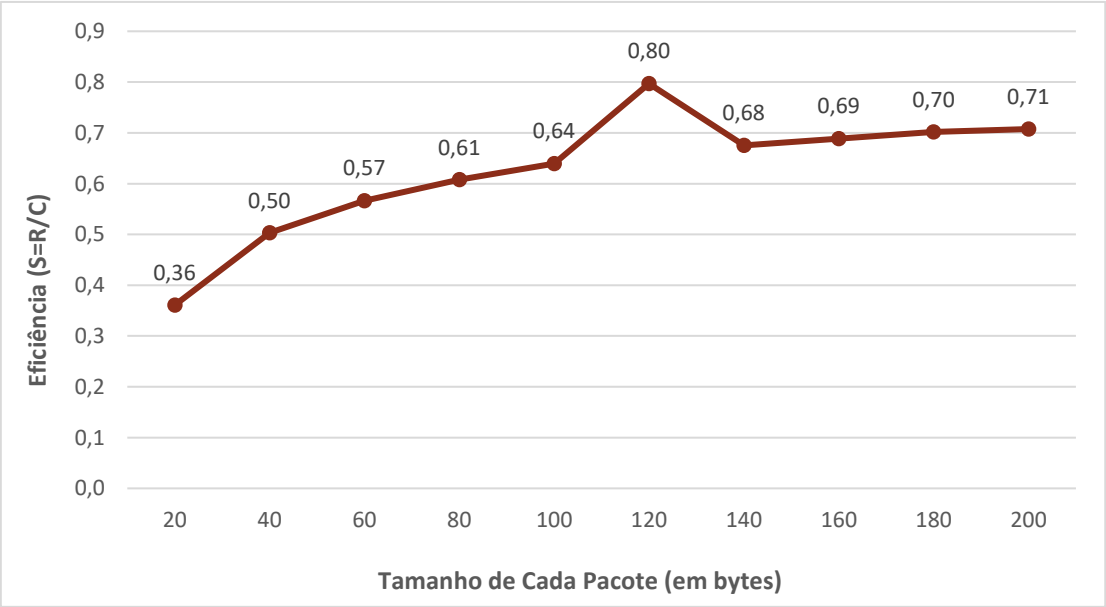
Variação do FER

Com este gráfico podemos concluir que a geração de erros no *BCC1* e no *BCC2* tem grande impacto na eficiência do programa. Isto deve-se principalmente ao facto de que quando há erros no *BCC1* o recetor não responde o que faz com que o emissor espere um número previamente escolhido de segundos atrasando bastante a execução. Os erros no *BCC2* não têm tanto efeito pois estes só causam o reenvio da trama, e este é imediato.



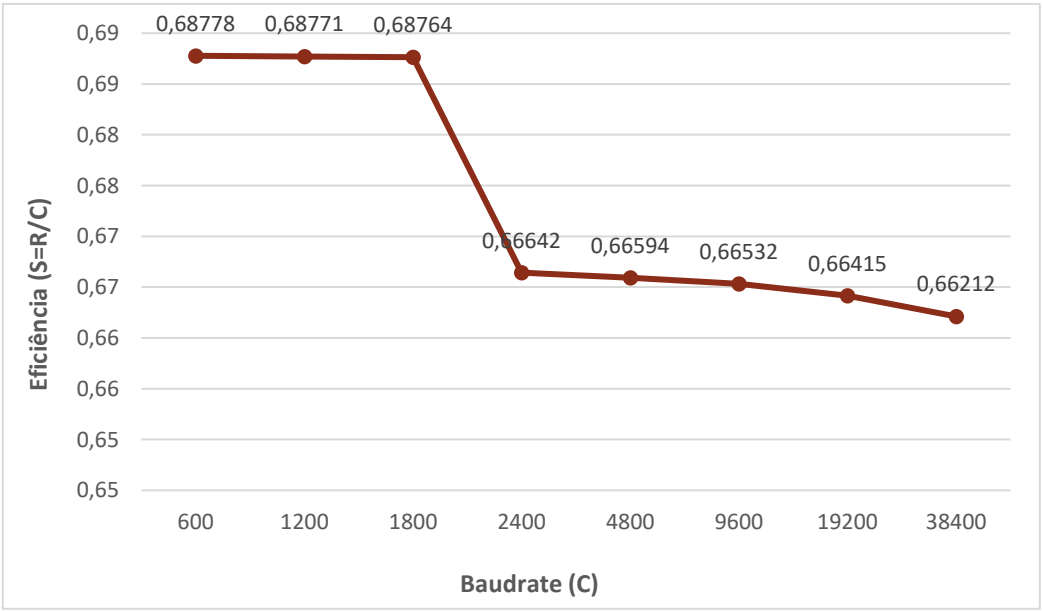
Variação do tamanho das tramas I

Com o gráfico seguinte podemos confirmar que quanto maior o tamanho de cada pacote, mais eficiente é a aplicação. Isto é porque é mandada mais informação de uma vez o que faz com menos tramas sejam mandadas e que o programa execute mais rapidamente.



Variação da capacidade da ligação (C)

Com este gráfico podemos concluir que com o aumento da capacidade de ligação, diminui a eficiência.



Em relação ao protocolo *Stop & Wait*, após a transmissão de um *packet* de informação, o emissor espera por uma confirmação positiva por parte do recetor, denominada por *acknowledgment*, **ACK**. Quando o recetor recebe o *packet*, caso não tenha nenhum erro, confirma com **ACK**, caso tenha erro, envia **NACK**. Assim que o emissor recebe a resposta do recetor, no caso de **ACK**, continua e envia um novo *packet*, mas no caso de **NACK**, volta a enviar o mesmo *packet*.

Na nossa aplicação foi usado um protocolo baseado no protocolo *Stop and Wait* para controlo de erros. Quando o emissor manda qualquer tipo de tramas (U, S ou I) espera uma resposta. Essa resposta é **RR** caso o recetor receba os dados sem erros, e **REJ** caso contrário. Assim, o emissor sabe se deve mandar uma nova trama ou reenviar a mesma. O *Nr* destas tramas de resposta varia conforme o emissor tenha enviado uma trama de *Ns* 0 ou 1, para este, no futuro, saber que trama deve mandar e para ajudar no tratamento de duplicados.

	Resposta do Recetor	
Trama enviada pelo Emissor	**Sem erros**	**Com erros**
Ns=0	RR (Nr=1)	REJ (Nr=0)
Ns=1	RR (Nr=0)	REJ (Nr=1)

Conclusões

O tema deste trabalho é o protocolo de ligação de dados, que consiste em fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio de transmissão, neste caso, um cabo série.

Adicionalmente, foi dado a conhecer o termo **independência entre camadas**, e cada um dos blocos funcionais da arquitetura da aplicação desenvolvida, *writer* e *reader*, cumpre esta independência. Na camada de ligação de dados não é feito qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de Informação. No que respeita a camada de aplicação, esta não conhece os detalhes do protocolo de ligação de dados, mas apenas a forma como o serviço é acedido.

Em suma, o trabalho foi concluído com sucesso, tendo-se cumprido todos os objetivos, e a sua elaboração contribuiu positivamente para um aprofundamento do conhecimento, tanto teórico como prático, do tema em questão.

Anexo I

writer.h:

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

#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

#define NUMMAX 3
#define TIMEOUT 3
#define sizePacketConst 100
#define bcc1ErrorPercentage 0
#define bcc2ErrorPercentage 0

#define FLAG 0x7E
#define A 0x03
#define SET_BCC (A ^ SET_C)
#define UA_BCC (A ^ UA_C)
#define UA_C 0x07
#define SET_C 0x03
#define C10 0x00
#define C11 0x40
#define C2Start 0x02
#define C2End 0x03
#define CRR0 0x05
#define CRR1 0x85
#define CREJ0 0x01
#define CREJ1 0x81
#define DISC 0x0B
#define headerC 0x01

#define Escape 0x7D
#define escapeFlag 0x5E
#define escapeEscape 0x5D
```

```c
#define T1 0x00
#define T2 0x01
#define L1 0x04
#define L2 0x0B


/*-------------------------Data Link Layer -------------------------*/

/*
*Envia trama de supervisão SET e recebe trama UA.
*Data link Layer
*/
int LLOPEN(int fd, int x);

/*
* Realiza stuffing das tramas I e envia-as.
* Data link layer
*/
int LLWRITE(int fd, unsigned char *mensagem, int size);

/*
* Envia trama de supervisão DISC, recebe DISC e envia UA.
* Data link layer
*/
void LLCLOSE(int fd);

/*
* Verifica se o UA foi recebido (com alarme).
* Data link layer
*/
void stateMachineUA(int *state, unsigned char *c);

/*
* Espera por uma trama de supervisão e retorna o seu C.
* Data link layer
*/
unsigned char readControlMessageC(int fd);

/*
* Envia uma trama de supervisão, sendo o C recebido como argumento
* da função a diferença de cada trama enviada.
* Data link layer
*/
void sendControlMessage(int fd, unsigned char C);

/*
* Calcula o valor do BCC2 de uma mensagem.
* Data link Layer
*/
```

```c
unsigned char calculoBCC2(unsigned char *mensagem, int size);

/*
* realiza o stuffing do BCC2.
* Data link layer
*/
unsigned char *stuffingBCC2(unsigned char BCC2, int *sizeBCC2);

/*
* Geração aleatória de erros no BCC1.
* Data link layer
*/
unsigned char *messUpBCC1(unsigned char *packet, int sizePacket);

/*
* Geração aleatória de erros no BCC2.
* Data link layer
*/
unsigned char *messUpBCC2(unsigned char *packet, int sizePacket);

/*-------------------------Application Link Layer ----------------------
----*/

/*
* Base da camada de aplicação pois é esta que controla todo o processo
* que ocorre nesta camada e que faz as chamadas às funções da camada
* de ligação.
* Application layer
*/
int main(int argc, char **argv);

/*
* Cria os pacotes de controlo START e END.
* Application layer
*/
unsigned char *controlPackageI(unsigned char state, off_t sizeFile,
unsigned char *fileName, int sizeOfFilename, int *sizeControlPackageI);


/*
* Abre um ficheiro e le o seu conteúdo.
* Application layer
*/
unsigned char *openReadFile(unsigned char *fileName, off_t *sizeFile);

/*
* Acrescenta o cabeçalho do nível de aplicação às tramas.
* Application layer
*/
```

```
unsigned char *headerAL(unsigned char *mensagem, off_t sizeFile, int
*sizePacket);

/*
* Divide uma mensagem proveniente do ficheiro em packets.
* Application layer
*/
unsigned char *splitMessage(unsigned char *mensagem, off_t *indice, int
*sizePacket, off_t sizeFile);
```

writer.c:

```
/*Non-Canonical Input Processing*/
#include "writer.h"

int sumAlarms = 0;
int flagAlarm = FALSE;
int trama = 0;
int paragem = FALSE;
unsigned char numMensagens = 0;
int numTotalTramas = 0;

struct termios oldtio, newtio;

//handler do sinal de alarme
void alarmHandler()
{
  printf("Alarm=%d\n", sumAlarms + 1);
  flagAlarm = TRUE;
  sumAlarms++;
}

int main(int argc, char **argv)
{
  int fd;
  off_t sizeFile; //tamanho do ficheiro em bytes
  off_t indice = 0;
  int sizeControlPackageI = 0;

  if ((argc < 3) ||
      ((strcmp("/dev/ttyS0", argv[1]) != 0) &&
       (strcmp("/dev/ttyS1", argv[1]) != 0)))
  {
    printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
    exit(1);
  }
  /*
```

```
  Open serial port device for reading and writing and not as controlling
tty
  because we don't want to get killed if linenoise sends CTRL-C.
  */
  fd = open(argv[1], O_RDWR | O_NOCTTY);
  if (fd < 0)
  {
    perror(argv[1]);
    exit(-1);
  }


  // instalar handler do alarme
  (void)signal(SIGALRM, alarmHandler);


  unsigned char *mensagem = openReadFile((unsigned char *)argv[2],
&sizeFile);

  //inicio do relógio
  struct timespec requestStart, requestEnd;
  clock_gettime(CLOCK_REALTIME, &requestStart);

  //se nao conseguirmos efetuar a ligaçao atraves do set e do ua o
programa termina
  if (!LLOPEN(fd, 0))
  {
    return -1;
  }

  int sizeOfFileName = strlen(argv[2]);
  unsigned char *fileName = (unsigned char *)malloc(sizeOfFileName);
  fileName = (unsigned char *)argv[2];
  unsigned char *start = controlPackageI(C2Start, sizeFile, fileName,
sizeOfFileName, &sizeControlPackageI);

  LLWRITE(fd, start, sizeControlPackageI);
  printf("Mandou trama START\n");

  int sizePacket = sizePacketConst;
  srand(time(NULL));

  while (sizePacket == sizePacketConst && indice < sizeFile)
  {
    //split mensagem
    unsigned char *packet = splitMessage(mensagem, &indice, &sizePacket,
sizeFile);
    printf("Mandou packet numero %d\n", numTotalTramas);
    //header nivel aplicação
```

```
    int headerSize = sizePacket;
    unsigned char *mensagemHeader = headerAL(packet, sizeFile,
&headerSize);
    //envia a mensagem
    if (!LLWRITE(fd, mensagemHeader, headerSize))
    {
      printf("Limite de alarmes atingido\n");
      return -1;
    }
  }

  unsigned char *end = controlPackageI(C2End, sizeFile, fileName,
sizeOfFileName, &sizeControlPackageI);
  LLWRITE(fd, end, sizeControlPackageI);
  printf("Mandou trama END\n");

  LLCLOSE(fd);

  //fim do relógio
  clock_gettime(CLOCK_REALTIME, &requestEnd);

  double accum = (requestEnd.tv_sec - requestStart.tv_sec) +
(requestEnd.tv_nsec - requestStart.tv_nsec) / 1E9;

  printf("Seconds passed: %f\n", accum);

  sleep(1);


  close(fd);
  return 0;
}

unsigned char *headerAL(unsigned char *mensagem, off_t sizeFile, int
*sizePacket)
{
  unsigned char *mensagemFinal = (unsigned char *)malloc(sizeFile + 4);
  mensagemFinal[0] = headerC;
  mensagemFinal[1] = numMensagens % 255;
  mensagemFinal[2] = (int)sizeFile / 256;
  mensagemFinal[3] = (int)sizeFile % 256;
  memcpy(mensagemFinal + 4, mensagem, *sizePacket);
  *sizePacket += 4;
  numMensagens++;
  numTotalTramas++;
  return mensagemFinal;
}
```

```
unsigned char *splitMessage(unsigned char *mensagem, off_t *indice, int
*sizePacket, off_t sizeFile)
{
  unsigned char *packet;
  int i = 0;
  off_t j = *indice;
  if (*indice + *sizePacket > sizeFile)
  {
    *sizePacket = sizeFile - *indice;
  }
  packet = (unsigned char *)malloc(*sizePacket);
  for (; i < *sizePacket; i++, j++)
  {
    packet[i] = mensagem[j];
  }
  *indice = j;
  return packet;
}

void stateMachineUA(int *state, unsigned char *c)
{

  switch (*state)
  {
  //recebe flag
  case 0:
    if (*c == FLAG)
      *state = 1;
    break;
  //recebe A
  case 1:
    if (*c == A)
      *state = 2;
    else
    {
      if (*c == FLAG)
        *state = 1;
      else
        *state = 0;
    }
    break;
  //recebe C
  case 2:
    if (*c == UA_C)
      *state = 3;
    else
    {
      if (*c == FLAG)
        *state = 1;
```

```
      else
        *state = 0;
    }
    break;
  //recebe BCC
  case 3:
    if (*c == UA_BCC)
      *state = 4;
    else
      *state = 0;
    break;
  //recebe FLAG final
  case 4:
    if (*c == FLAG)
    {
      paragem = TRUE;
      alarm(0);
      printf("Recebeu UA\n");
    }
    else
      *state = 0;
    break;
  }
}

int LLOPEN(int fd, int x)
{

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

  bzero(&newtio, sizeof(newtio));
  newtio.c_cflag = 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] = 1; /* inter-unsigned character timer unused */
  newtio.c_cc[VMIN] = 0;  /* blocking read until 5 unsigned 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);
  }

  printf("New termios structure set\n");


  unsigned char c;
  do
  {
    sendControlMessage(fd, SET_C);
    alarm(TIMEOUT);
    flagAlarm = 0;
    int state = 0;

    while (!paragem && !flagAlarm)
    {
      read(fd, &c, 1);
      stateMachineUA(&state, &c);
    }
  } while (flagAlarm && sumAlarms < NUMMAX);
  printf("flag alarm %d\n", flagAlarm);
  printf("soma %d\n", sumAlarms);
  if (flagAlarm && sumAlarms == 3)
  {
    return FALSE;
  }
  else
  {
    flagAlarm = FALSE;
    sumAlarms = 0;
    return TRUE;
  }
}

int LLWRITE(int fd, unsigned char *mensagem, int size)
{
  unsigned char BCC2;
```

```
  unsigned char *BCC2Stuffed = (unsigned char *)malloc(sizeof(unsigned
char));
  unsigned char *mensagemFinal = (unsigned char *)malloc((size + 6) *
sizeof(unsigned char));
  int sizeMensagemFinal = size + 6;
  int sizeBCC2 = 1;
  BCC2 = calculoBCC2(mensagem, size);
  BCC2Stuffed = stuffingBCC2(BCC2, &sizeBCC2);
  int rejeitado = FALSE;

  mensagemFinal[0] = FLAG;
  mensagemFinal[1] = A;
  if (trama == 0)
  {
    mensagemFinal[2] = C10;
  }
  else
  {
    mensagemFinal[2] = C11;
  }
  mensagemFinal[3] = (mensagemFinal[1] ^ mensagemFinal[2]);

  int i = 0;
  int j = 4;
  for (; i < size; i++)
  {
    if (mensagem[i] == FLAG)
    {
      mensagemFinal = (unsigned char *)realloc(mensagemFinal,
++sizeMensagemFinal);
      mensagemFinal[j] = Escape;
      mensagemFinal[j + 1] = escapeFlag;
      j = j + 2;
    }
    else
    {
      if (mensagem[i] == Escape)
      {
        mensagemFinal = (unsigned char *)realloc(mensagemFinal,
++sizeMensagemFinal);
        mensagemFinal[j] = Escape;
        mensagemFinal[j + 1] = escapeEscape;
        j = j + 2;
      }
      else
      {
        mensagemFinal[j] = mensagem[i];
        j++;
      }
```

```
    }
  }

  if (sizeBCC2 == 1)
    mensagemFinal[j] = BCC2;
  else
  {
    mensagemFinal = (unsigned char *)realloc(mensagemFinal,
++sizeMensagemFinal);
    mensagemFinal[j] = BCC2Stuffed[0];
    mensagemFinal[j + 1] = BCC2Stuffed[1];
    j++;
  }
  mensagemFinal[j + 1] = FLAG;

  //mandar mensagem
  do
  {

    unsigned char *copia;
    copia = messUpBCC1(mensagemFinal, sizeMensagemFinal); //altera bcc1
    copia = messUpBCC2(copia, sizeMensagemFinal);         //altera bcc2
    write(fd, copia, sizeMensagemFinal);

    flagAlarm = FALSE;
    alarm(TIMEOUT);
    unsigned char C = readControlMessageC(fd);
    if ((C == CRR1 && trama == 0) || (C == CRR0 && trama == 1))
    {
      printf("Recebeu rr %x, trama = %d\n", C, trama);
      rejeitado = FALSE;
      sumAlarms = 0;
      trama ^= 1;
      alarm(0);
    }
    else
    {
      if (C == CREJ1 || C == CREJ0)
      {
        rejeitado = TRUE;
        printf("Recebeu rej %x, trama=%d\n", C, trama);
        alarm(0);
      }
    }
  } while ((flagAlarm && sumAlarms < NUMMAX) || rejeitado);
  if (sumAlarms >= NUMMAX)
    return FALSE;
  else
    return TRUE;
```

```
}

void sendControlMessage(int fd, unsigned char C)
{
  unsigned char message[5];
  message[0] = FLAG;
  message[1] = A;
  message[2] = C;
  message[3] = message[1] ^ message[2];
  message[4] = FLAG;
  write(fd, message, 5);
}

unsigned char readControlMessageC(int fd)
{
  int state = 0;
  unsigned char c;
  unsigned char C;

  while (!flagAlarm && state != 5)
  {
    read(fd, &c, 1);
    switch (state)
    {
    //recebe FLAG
    case 0:
      if (c == FLAG)
        state = 1;
      break;
    //recebe A
    case 1:
      if (c == A)
        state = 2;
      else
      {
        if (c == FLAG)
          state = 1;
        else
          state = 0;
      }
      break;
    //recebe c
    case 2:
      if (c == CRR0 || c == CRR1 || c == CREJ0 || c == CREJ1 || c ==
DISC)
      {
        C = c;
        state = 3;
      }
```

```
      else
      {
        if (c == FLAG)
          state = 1;
        else
          state = 0;
      }
      break;
    //recebe BCC
    case 3:
      if (c == (A ^ C))
        state = 4;
      else
        state = 0;
      break;
    //recebe FLAG final
    case 4:
      if (c == FLAG)
      {
        alarm(0);
        state = 5;
        return C;
      }
      else
        state = 0;
      break;
    }
  }
  return 0xFF;
}

unsigned char calculoBCC2(unsigned char *mensagem, int size)
{
  unsigned char BCC2 = mensagem[0];
  int i;
  for (i = 1; i < size; i++)
  {
    BCC2 ^= mensagem[i];
  }
  return BCC2;
}

unsigned char *stuffingBCC2(unsigned char BCC2, int *sizeBCC2)
{
  unsigned char *BCC2Stuffed;
  if (BCC2 == FLAG)
  {
    BCC2Stuffed = (unsigned char *)malloc(2 * sizeof(unsigned char *));
    BCC2Stuffed[0] = Escape;
```

```
    BCC2Stuffed[1] = escapeFlag;
    (*sizeBCC2)++;
  }
  else
  {
    if (BCC2 == Escape)
    {
      BCC2Stuffed = (unsigned char *)malloc(2 * sizeof(unsigned char *));
      BCC2Stuffed[0] = Escape;
      BCC2Stuffed[1] = escapeEscape;
      (*sizeBCC2)++;
    }
  }

  return BCC2Stuffed;
}

unsigned char *openReadFile(unsigned char *fileName, off_t *sizeFile)
{
  FILE *f;
  struct stat metadata;
  unsigned char *fileData;

  if ((f = fopen((char *)fileName, "rb")) == NULL)
  {
    perror("error opening file!");
    exit(-1);
  }
  stat((char *)fileName, &metadata);
  (*sizeFile) = metadata.st_size;
  printf("This file has %ld bytes \n", *sizeFile);

  fileData = (unsigned char *)malloc(*sizeFile);

  fread(fileData, sizeof(unsigned char), *sizeFile, f);
  return fileData;
}

unsigned char *controlPackageI(unsigned char state, off_t sizeFile,
unsigned char *fileName, int sizeOfFileName, int *sizeControlPackageI)
{
  *sizeControlPackageI = 9 * sizeof(unsigned char) + sizeOfFileName;
  unsigned char *package = (unsigned char *)malloc(*sizeControlPackageI);

  if (state == C2Start)
    package[0] = C2Start;
  else
    package[0] = C2End;
  package[1] = T1;
```

```
  package[2] = L1;
  package[3] = (sizeFile >> 24) & 0xFF;
  package[4] = (sizeFile >> 16) & 0xFF;
  package[5] = (sizeFile >> 8) & 0xFF;
  package[6] = sizeFile & 0xFF;
  package[7] = T2;
  package[8] = sizeOfFileName;
  int k = 0;
  for (; k < sizeOfFileName; k++)
  {
    package[9 + k] = fileName[k];
  }
  return package;
}

void LLCLOSE(int fd)
{
  sendControlMessage(fd, DISC);
  printf("Mandou DISC\n");
  unsigned char C;
  //espera ler o DISC
  C = readControlMessageC(fd);
  while (C != DISC)
  {
    C = readControlMessageC(fd);
  }
  printf("Leu DISC\n");
  sendControlMessage(fd, UA_C);
  printf("Mandou UA final\n");
  printf("Writer terminated \n");

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

unsigned char *messUpBCC2(unsigned char *packet, int sizePacket)
{
  unsigned char *copia = (unsigned char *)malloc(sizePacket);
  memcpy(copia, packet, sizePacket);
  int r = (rand() % 100) + 1;
  if (r <= bcc2ErrorPercentage)
  {
    int i = (rand() % (sizePacket - 5)) + 4;
    unsigned char randomLetter = (unsigned char)('A' + (rand() % 26));
    copia[i] = randomLetter;
    printf("Modifiquei BCC2\n");
```

```
  }
  return copia;
}

unsigned char *messUpBCC1(unsigned char *packet, int sizePacket)
{
  unsigned char *copia = (unsigned char *)malloc(sizePacket);
  memcpy(copia, packet, sizePacket);
  int r = (rand() % 100) + 1;
  if (r <= bcc1ErrorPercentage)
  {
    int i = (rand() % 3) + 1;
    unsigned char randomLetter = (unsigned char)('A' + (rand() % 26));
    copia[i] = randomLetter;
    printf("Modifiquei BCC1\n");
  }
  return copia;
}
```

reader.h:

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

#define BAUDRATE B38400
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

#define FLAG 0x7E
#define A 0x03
#define SET_BCC A ^ SET_C
#define UA_BCC A ^ UA_C
#define SET_C 0x03
#define UA_C 0x07
#define C10 0x00
#define C11 0x40
#define RR_C0 0x05
#define RR_C1 0x85
#define REJ_C0 0x01
```

```c
#define REJ_C1 0x81
#define DISC_C 0x0B
#define C2End 0x03

#define Escape 0x7D
#define escapeFlag 0x5E
#define escapeEscape 0x5D

/*-------------------------Data Link Layer -------------------------*/

/*
* Lê trama de controlo SET e envia a trama UA.
* Data link layer
*/
void LLOPEN(int fd);

/*
* Lê tramas I e faz destuffing.
* Data link layer
*/
unsigned char *LLREAD(int fd, int *sizeMessage);

/*
* Lê trama de controlo DISC, envia DISC de volta e recebe UA.
* Data link Layer
*/
void LLCLOSE(int fd);

/*
* Ciclo de leitura que quebra após ler uma trama de controlo C
* que seja igual ao C recebido como argumento da função.
* Data link layer
*/
int readControlMessage(int fd, unsigned char C);

/*
* Envia uma trama de controlo, sendo o C recebido como argumento da
função a diferença de cada trama enviada.
* Data link layer layer
*/
void sendControlMessage(int fd, unsigned char C);

/*
* Verifica se o BCC2 recebido na mensagem está correto.
* Data link layer
*/
int checkBCC2(unsigned char *message, int sizeMessage);
```

```c
/*-------------------------Application Link Layer ---------------------
----*/

/*
* Base da camada de aplicação pois é esta que controla todo o processo
* que ocorre nesta camada e que faz as chamadas às funções da camada
* de ligação.
* Application layer
*/
int main(int argc, char **argv);


/*
* Obtém nome do ficheiro a partir da trama START.
* Application layer
*/
unsigned char *nameOfFileFromStart(unsigned char *start);

/*
* Obtém tamanho do ficheiro a partir da trama START.
* Application layer
*/
off_t sizeOfFileFromStart(unsigned char *start);

/*
* Remove o cabeçalho do nível de aplicação das tramas I.
* Application layer
*/
unsigned char *removeHeader(unsigned char *toRemove, int sizeToRemove,
int *sizeRemoved);

/*
* Verifica se a trama recebida e a trama END.
* Application layer
*/
int isEndMessage(unsigned char *start, int sizeStart, unsigned char *end,
int sizeEnd);

/*
* Cria ficheiro com os dados recebidos nas tramas I.
* Application layer
*/
void createFile(unsigned char *mensagem, off_t *sizeFile, unsigned char
*filename);
```

reader.c:

```c
/*Non-Canonical Input Processing*/
#include "reader.h"

int esperado = 0;
struct termios oldtio, newtio;

int main(int argc, char **argv)
{
  int fd;
  int sizeMessage = 0;
  unsigned char *mensagemPronta;
  int sizeOfStart = 0;
  unsigned char *start;
  off_t sizeOfGiant = 0;
  unsigned char *giant;
  off_t index = 0;

  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);
  }
  /*
    Open serial port device for reading and writing and not as
controlling tty
    because we don't want to get killed if linenoise sends CTRL-C.
  */
  fd = open(argv[1], O_RDWR | O_NOCTTY);
  if (fd < 0)
  {
    perror(argv[1]);
    exit(-1);
  }


  LLOPEN(fd);
  start = LLREAD(fd, &sizeOfStart);

  unsigned char *nameOfFile = nameOfFileFromStart(start);
  sizeOfGiant = sizeOfFileFromStart(start);

  giant = (unsigned char *)malloc(sizeOfGiant);

  while (TRUE)
  {
```

```
    mensagemPronta = LLREAD(fd, &sizeMessage);
    if (sizeMessage == 0)
      continue;
    if (isEndMessage(start, sizeOfStart, mensagemPronta, sizeMessage))
    {
      printf("End message received\n");
      break;
    }

    int sizeWithoutHeader = 0;

    mensagemPronta = removeHeader(mensagemPronta, sizeMessage,
&sizeWithoutHeader);

    memcpy(giant + index, mensagemPronta, sizeWithoutHeader);
    index += sizeWithoutHeader;
  }

  printf("Mensagem: \n");
  int i = 0;
  for (; i < sizeOfGiant; i++)
  {
    printf("%x", giant[i]);
  }

  createFile(giant, &sizeOfGiant, nameOfFile);

  LLCLOSE(fd);

  sleep(1);

  close(fd);
  return 0;
}

void LLCLOSE(int fd)
{
  readControlMessage(fd, DISC_C);
  printf("Recebeu DISC\n");
  sendControlMessage(fd, DISC_C);
  printf("Mandou DISC\n");
  readControlMessage(fd, UA_C);
  printf("Recebeu UA\n");
  printf("Receiver terminated\n");

  tcsetattr(fd, TCSANOW, &oldtio);
}

void LLOPEN(int fd)
```

```
{

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

  bzero(&newtio, sizeof(newtio));
  newtio.c_cflag = 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] = 1; /* inter-character timer unused */
  newtio.c_cc[VMIN] = 0;  /* blocking read until 5 chars received */

  /*
    VTIME e VMIN devem ser alterados de forma a proteger com um
temporizador a
    leitura do(s) próximo(s) caracter(es)
  */

  tcflush(fd, TCIOFLUSH);

  printf("New termios structure set\n");

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

  if (readControlMessage(fd, SET_C))
  {
    printf("Recebeu SET\n");
    sendControlMessage(fd, UA_C);
    printf("Mandou UA\n");
  }
}

unsigned char *LLREAD(int fd, int *sizeMessage)
{
  unsigned char *message = (unsigned char *)malloc(0);
  *sizeMessage = 0;
  unsigned char c_read;
  int trama = 0;
```

```
int mandarDados = FALSE;
unsigned char c;
int state = 0;

while (state != 6)
{
  read(fd, &c, 1);
  //printf("%x\n",c);
  switch (state)
  {
  //recebe flag
  case 0:
    if (c == FLAG)
      state = 1;
    break;
  //recebe A
  case 1:
    //printf("1state\n");
    if (c == A)
      state = 2;
    else
    {
      if (c == FLAG)
        state = 1;
      else
        state = 0;
    }
    break;
  //recebe C
  case 2:
    //printf("2state\n");
    if (c == C10)
    {
      trama = 0;
      c_read = c;
      state = 3;
    }
    else if (c == C11)
    {
      trama = 1;
      c_read = c;
      state = 3;
    }
    else
    {
      if (c == FLAG)
        state = 1;
      else
        state = 0;
```

```
    }
    break;
  //recebe BCC
  case 3:
    //printf("3state\n");
    if (c == (A ^ c_read))
      state = 4;
    else
      state = 0;
    break;
  //recebe FLAG final
  case 4:
    //printf("4state\n");
    if (c == FLAG)
    {
      if (checkBCC2(message, *sizeMessage))
      {
        if (trama == 0)
          sendControlMessage(fd, RR_C1);
        else
          sendControlMessage(fd, RR_C0);

        state = 6;
        mandarDados = TRUE;
        printf("Enviou RR, T: %d\n", trama);
      }
      else
      {
        if (trama == 0)
          sendControlMessage(fd, REJ_C1);
        else
          sendControlMessage(fd, REJ_C0);
        state = 6;
        mandarDados = FALSE;
        printf("Enviou REJ, T: %d\n", trama);
      }
    }
    else if (c == Escape)
    {
      state = 5;
    }
    else
    {
      message = (unsigned char *)realloc(message, ++(*sizeMessage));
      message[*sizeMessage - 1] = c;
    }
    break;
  case 5:
    //printf("5state\n");
```

```
      if (c == escapeFlag)
      {
        message = (unsigned char *)realloc(message, ++(*sizeMessage));
        message[*sizeMessage - 1] = FLAG;
      }
      else
      {
        if (c == escapeEscape)
        {
          message = (unsigned char *)realloc(message, ++(*sizeMessage));
          message[*sizeMessage - 1] = Escape;
        }
        else
        {
          perror("Non valid character after escape character");
          exit(-1);
        }
      }
      state = 4;
      break;
    }
  }
  printf("Message size: %d\n", *sizeMessage);
  //message tem BCC2 no fim
  message = (unsigned char *)realloc(message, *sizeMessage - 1);

  *sizeMessage = *sizeMessage - 1;
  if (mandarDados)
  {
    if (trama == esperado)
    {
      esperado ^= 1;
    }
    else
      *sizeMessage = 0;
  }
  else
    *sizeMessage = 0;
  return message;
}

int checkBCC2(unsigned char *message, int sizeMessage)
{
  int i = 1;
  unsigned char BCC2 = message[0];
  for (; i < sizeMessage - 1; i++)
  {
    BCC2 ^= message[i];
  }
```

```
  if (BCC2 == message[sizeMessage - 1])
  {
    return TRUE;
  }
  else
    return FALSE;
}

void sendControlMessage(int fd, unsigned char C)
{
  unsigned char message[5];
  message[0] = FLAG;
  message[1] = A;
  message[2] = C;
  message[3] = message[1] ^ message[2];
  message[4] = FLAG;
  write(fd, message, 5);
}

int readControlMessage(int fd, unsigned char C)
{
  int state = 0;
  unsigned char c;

  while (state != 5)
  {
    read(fd, &c, 1);
    switch (state)
    {
    //recebe flag
    case 0:
      if (c == FLAG)
        state = 1;
      break;
    //recebe A
    case 1:
      if (c == A)
        state = 2;
      else
      {
        if (c == FLAG)
          state = 1;
        else
          state = 0;
      }
      break;
    //recebe C
    case 2:
      if (c == C)
```

```
        state = 3;
      else
      {
        if (c == FLAG)
          state = 1;
        else
          state = 0;
      }
      break;
    //recebe BCC
    case 3:
      if (c == (A ^ C))
        state = 4;
      else
        state = 0;
      break;
    //recebe FLAG final
    case 4:
      if (c == FLAG)
      {
        //printf("Recebeu mensagem\n");
        state = 5;
      }
      else
        state = 0;
      break;
    }
  }
  return TRUE;
}

unsigned char *removeHeader(unsigned char *toRemove, int sizeToRemove,
int *sizeRemoved)
{
  int i = 0;
  int j = 4;
  unsigned char *messageRemovedHeader = (unsigned char
*)malloc(sizeToRemove - 4);
  for (; i < sizeToRemove; i++, j++)
  {
    messageRemovedHeader[i] = toRemove[j];
  }
  *sizeRemoved = sizeToRemove - 4;
  return messageRemovedHeader;
}

int isEndMessage(unsigned char *start, int sizeStart, unsigned char *end,
int sizeEnd)
{
```

```
  int s = 1;
  int e = 1;
  if (sizeStart != sizeEnd)
    return FALSE;
  else
  {
    if (end[0] == C2End)
    {
      for (; s < sizeStart; s++, e++)
      {
        if (start[s] != end[e])
          return FALSE;
      }
      return TRUE;
    }
    else
    {
      return FALSE;
    }
  }
}

off_t sizeOfFileFromStart(unsigned char *start)
{
  return (start[3] << 24) | (start[4] << 16) | (start[5] << 8) |
(start[6]);
}

unsigned char *nameOfFileFromStart(unsigned char *start)
{

  int L2 = (int)start[8];
  unsigned char *name = (unsigned char *)malloc(L2 + 1);

  int i;
  for (i = 0; i < L2; i++)
  {
    name[i] = start[9 + i];
  }

  name[L2] = '\0';
  return name;
}

void createFile(unsigned char *mensagem, off_t *sizeFile, unsigned char
filename[])
{
  FILE *file = fopen((char *)filename, "wb+");
  fwrite((void *)mensagem, 1, *sizeFile, file);
```

```
  printf("%zd\n", *sizeFile);
  printf("New file created\n");
  fclose(file);
}
```

Anexo II

Nº total de bytes	10968
Nº total de bits	87744
C (Baudrate)	38400
Tamanho de cada pacote (bytes)	100

Probabilidade de erro (%bcc1 +%bcc2)	Tempo (s)	R (bits/tempo)	S (R/C)	S (R/C) Média
0+0	3,45	25424,30	0,66209	0,662150525
	3,45	25428,86	0,66221	
2+2	6,64	13217,15	0,34420	0,289357955
	9,74	9005,54	0,23452	
4+4	19,19	4572,48	0,11907	0,119273584
	19,13	4587,74	0,11947	
6+6	25,46	3447,01	0,08977	0,096271503
	22,23	3946,64	0,10278	
8+8	34,65	2532,46	0,06595	0,066073546
	34,52	2541,98	0,06620	
10+10	68,92	1273,12	0,03315	0,036785126
	56,54	1551,98	0,04042	

Figura 1: Tabela de cálculos da eficiência variando o número de erros.

Nº total de bytes	10968
Nº total de bits	87744
C (Baudrate)	38400

Tamanho de cada pacote (bytes)	Tempo (s)	R (bits/tempo)	S (R/C)	S (R/C) Média
20	6,33	13861,76	0,36098	0,360793096
	6,34	13847,15	0,36060	
40	4,54	19311,92	0,50291	0,503208649
	4,54	19334,50	0,50350	
60	4,03	21749,29	0,56639	0,566534629
	4,03	21760,57	0,56668	
80	3,76	23342,25	0,60787	0,607607357
	3,76	23322,00	0,60734	
100	3,57	24555,24	0,63946	0,639494004
	3,57	24557,90	0,63953	
120	2,45	35781,47	0,93181	0,796928878
	3,45	25422,67	0,66205	
140	3,38	25942,92	0,67560	0,675596765
	3,38	25942,91	0,67560	
160	3,32	26449,28	0,68878	0,688671523
	3,32	26440,69	0,68856	
180	3,25	26959,34	0,70207	0,702064222
	3,25	26959,19	0,70206	
200	3,23	27172,68	0,70762	0,707598807
	3,23	27170,91	0,70758	

Figura 2: Tabela de cálculos da eficiência variando o tamanho da trama I.

Nº total de bytes	10968
Nº total de bits	87744
Tamanho de cada pacote (bytes)	100

C (Baudrate)	Tempo (s)	R (bits/tempo)	S (R/C)	S (R/C) Média
600	212,63	412,66	0,68777	0,687781318
	212,62	412,68	0,68780	
1200	106,32	825,25	0,68771	0,687705854
	106,32	825,25	0,68771	
1800	70,90	1237,65	0,68758	0,687640463
	70,88	1237,86	0,68770	
2400	54,86	1599,41	0,66642	0,666424043
	54,86	1599,42	0,66643	
4800	27,45	3196,48	0,66593	0,665938021
	27,45	3196,53	0,66594	
9600	13,74	6387,13	0,66533	0,665317494
	13,74	6386,97	0,66531	
19200	6,88	12751,47	0,66414	0,664153831
	6,88	12752,04	0,66417	
38400	3,45	25425,85	0,66213	0,662119054
	3,45	25424,89	0,66211	

Figura 3: Tabela de cálculos da eficiência variando a capacidade de ligação.