U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Redes de Computadores

1º Trabalho Laboratorial

11 de novembro de 2018

Beatriz Garrido up201504710
Diogo Santos up201505606
João Costa up201404935

Índice

Sumário

No âmbito da unidade curricular de Redes de Computadores, este relatório tem como objetivo a implementação do primeiro trabalho laboratorial, referente à transferência de dados. Este trabalho consiste no desenvolvimento de uma aplicação capaz de enviar ficheiros de um computador para outro através de uma porta série.

Dito isto, o trabalho foi realizado com sucesso, visto que os objetivos propostos foram concretizados e a aplicação foi desenvolvida na perfeição, sem qualquer perda de dados.

Introdução

O objetivo deste trabalho é implementar um protocolo de ligação de dados, de acordo com as especificações descritas no guião fornecido, bem como testar o protocolo com uma aplicação simples de transferência de ficheiros, igualmente especificada.

Relativamente ao relatório, a sua função é explicar toda a lógica presente no trabalho, seguindo a seguinte estrutura:

- **Arquitetura:** exposição dos blocos funcionais e interfaces;
- **Estrutura do código:** identificação das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura;
- **Casos de uso principais:** identificação dos principais casos de uso e sequências de chamada de funções;
- **Protocolo de ligação lógica:** identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes, com apresentação de extratos de código;

- **Protocolo de aplicação:** identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes, com apresentação de extratos de código;

- **Validação:** descrição dos testes efetuados com apresentação quantificada dos resultados;

- **Elementos de valorização:** identificação dos pontos adicionais implementados e descrição da estratégia de implementação;

- **Conclusão:** síntese da informação apresentada nas secções anteriores e reflexão sobre os objetivos de aprendizagem alcançados;

Arquitetura

Este trabalho está dividido em duas principais camadas lógicas, com a finalidade de uma melhor organização do protocolo e aperfeiçoamento da aplicação.

Estrutura do código

Camada de Ligação

A camada de ligação (ou *Data Link Layer*) proporciona os meios necessários para a transferência de dados entre camadas. Pode também fornecer códigos de deteção de erros, cajo estes ocorram na camada física.

Assim, esta categoria contém as funções necessárias para o estabelecimento de ligação (LLOPEN), escrita e leitura na porta série (LLWRITE e LLREAD respetivamente), fecho da ligação (LLCLOSE) e deteção de erros, como já referido anteriormente.

Camada de Aplicação

A camada de aplicação (ou *Application Layer*) é a que se encontra diretamente acima da camada acima mencionada,

responsável pela comunicação entre o utilizador e a interface. É a partir desta que se inicia o processo de transferência de dados, sendo a que contém a função *main()*.

Casos de uso principais

A aplicação desenvolvida pode ser corrida de duas maneiras diferentes:

Como emissor

A transmissão de um ficheiro para outro computador necessita de 2 parâmetros. Depois da referente a porta série (normalmente /dev/ttyS0) é necessário introduzir o tipo de comunicação – "*sender*" – e o nome do ficheiro que se deseja enviar.

De notar que, se após um determinado número de *timeouts*, nenhuma ligação for estabelecida por parte do recetor, é invocada a função de fecho de ligação.

Como recetor

A receção do ficheiro enviado apenas necessita do parâmetro referente ao tipo de comunicação – "*receiver*" – depois da introdução da porta série.

Ao contrário do emissor, esta comunicação não tem tempo limitado, visto que fica constantemente à espera que o emissor inicie a ligação.

Protocolo de ligação lógica

O protocolo de ligação lógica é implementado na camada de ligação (*data link Layer),* da qual depende a camada de aplicação (A*pplication Layer*).

Nesta camada são implementadas as seguintes funcionalidades:

- Estabelecer e terminar uma ligação através da porta série, bem como transmitir e receber dados/mensagens na mesma.
- Criar e enviar comandos (tramas I, SET e DISC) e respostas (UA, RR e REJ).

- Receber mensagens através da porta série.
- Fazer *stuff* e *destuff* de pacotes da camada de aplicação.
- Criar o *packet header* e juntar ao pacote de dados, no caso do emissor, ou removê-lo no caso do recetor
- Verificar se a informação recebida é igual à envida, através do cabeçalho em todas as tramas. Nas tramas I, além do cabeçalho usa-se uma variável de controlo: BCC2.

LLOPEN

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

Função responsável por estabelecer a ligação através da porta série.

Quando esta função é chamada pelo emissor, este envia a trama SET e aguarda resposta do recetor, ou seja, a mensagem UA. Caso o recetor não receba uma resposta ao fim de 3 *timeouts,* cuja duração, até ser atingido, é definida através de um alarme, a ligação é fechada.

Quando esta função é chamada pelo recetor, a função fica à espera de receber a mensagem SET, sendo que, quando a recebe, envia como resposta a mensagem UA e a ligação é estabelecida. No caso do recetor não é implementado qualquer tipo de alarme, ficando este indefinidamente à espera de receber a mensagem SET.

LLWRITE

```
int LLWRITE(int fd, char *buffer, int lenght);
```

Função responsável por tentar escrever na porta série, os dados através da função *write()* e fica a aguardar uma resposta. Caso não receba uma resposta ao fim de 3 *timeouts*, a ligação é fechada. Quando a resposta for recebida e, caso esta contenha a variável de controlo RR, a mensagem foi transmitida corretamente.

Caso, a resposta contenha a variável de controlo REJ, a mensagem não foi transmitida corretamente, o que implica a mensagem ser retransmitida e o valor de *timeouts* permitido restaurado para o valor inicial.

LLREAD

```
int LLREAD(int fd, char *buffer);
```

Função responsável por tentar receber através da porta série os dados enviados.

Caso esta receba uma mensagem inválida, envia a variável de controlo REJ através da porta série. Caso a mensagem recebida for válida, a informação é guardada e é realizado o envio da variável de controlo RR através da porta série.

Caso receba a mensagem DISC, a ligação deve ser encerrada.

LLCLOSE

```
int LLCLOSE(int fd, int com_type);
```

Função responsável por terminar a ligação através da porta série.

Caso seja o emissor a chamar esta função, a mensagem DISC é enviada pela porta série e este fica a aguardar pela receção da trama DISC. Depois de o receber, envia a mensagem UA.

Caso seja o recetor a chamar a função, aguarda pela receção da trama DISC e quando o receber reenvia-o e fica a aguardar pela receção da mensagem UA.

Quando estes eventos ocorrem, a ligação através da porta série é terminada.

Stuffing

```
char* stuffing(char* payload, int* length);
```

Função responsável pelo mecanismo de *byte stuffing* da transparência de dados.

Caso os dados enviados contenham carateres 0x7E estes são substituídos por 0x7D 0x5E. Ou por 0x7D 0x5D, caso sejam carateres 0x7D.

Além disso, o cálculo de BCC2 é feito antes da operação de *stuffing*.

Destuffing

```
char* destuffing(char* msg, int* length);
```

Função responsável pelo mecanismo de *byte destuffing* da transparência de dados.

Caso os dados recebidos contenham os carateres 0x7D 0x5E de forma consecutiva, estes são substituídos por 0x7E. Ou então, por 0x7D caso sejam recebidos os carateres 0x7D 0x5D.

Além disso, o cálculo de BCC2 é feito depois da operação de *destuffing*.

Verify_bcc2

```
char* verify_bcc2(char* control_message, int* length);
```

Função responsável pela comparação da variável de controlo BCC2 recebida com a enviada, que permite saber se os dados foram enviados sem erros, caso estas duas sejam iguais.

Control_frame

```
char* control_frame(char* filename, FILE *file, int start,
int* frame_size);
```

Função responsável pela criação do pacote de controlo START/END.

Get_info

```
char* get_info(char* control, int* file_size);
```

Função responsável pela extração do nome e tamanho do ficheiro a ser transferido.

Header

```
char* header(char* buffer, int* length, short
sequence_number);
```

Função responsável pela adição do campo de controlo, número de sequência, e tamanho da *payload* às tramas que vão ser enviados pelo emissor.

Remove_header

```
char* remove_header(char* buffer, int* length);
```

Função responsável pela remoção do campo de controlo, número de sequência, e tamanho da *payload,* das tramas que vão ser recebidos pelo recetor.

Protocolo de aplicação

O protocolo de aplicação é implementado na camada de aplicação (“*Application Layer”)* que é a camada de mais alto nível, implementada neste projeto e é responsável pelas seguintes funcionalidades:

- Envio e receção de pacotes de controlo;
- Envio e receção de pacotes de dados;
- Envio e receção do ficheiro definido.

Variáveis do Nível de Aplicação

- **com_type:** indica se a aplicação está a correr no modo de *sender* ou *receiver*.
- **chunk_size:** indica o tamanho dos fragmentos de informação que vão ser enviados através da porta série de cada vez.

Emissor

O protocolo de aplicação implementado para o modo de emissor, consiste em:

- Abertura do ficheiro em modo de leitura de modo a ser possível usar a informação deste.
- Estabelecimento de comunicação através da função *LLOPEN().*
- Envio da trama de controlo inicial obtida através da função *control_frame().*

- Envio de fragmentos do ficheiro com tamanho definido por "chunk_size", com o devido campo de controlo, número de sequência e tamanho obtidos na função *header()* e com transparência assegurada pela função *stuffing()*.
- Criação de uma barra de progresso que indica a percentagem de informação que está a ser enviada em relação à informação total.
- Envio da trama de controlo final obtida através da função *control_frame()*.
- Impressão, no terminal, do tempo de envio e do débito.

Recetor

O protocolo de aplicação implementado para o modo de recetor, consiste em:

- Estabelecimento de comunicação através da função *LLOPEN()*.
- Leitura da informação do ficheiro através da função *LLREAD()*.
- Verificação se a informação recebida é a mesma do que a enviada.
- Caso seja válida, envia RR e começa a escrita num ficheiro com o mesmo nome do ficheiro enviado.
- Caso não seja válida, envia RR e não realiza qualquer escrita no ficheiro.
- Ocorrência de *destuffing* e verificação do BCC2.
- Invocação da função *remove_header(),* que remove o campo de controlo, número de sequência e tamanho da informação recebida.
- Criação de uma barra de progresso que indica a percentagem de informação que está a ser recebida em relação à informação total.
- Impressão no terminal o tempo de envio e o débito binário.

Validação

Nesta secção abordam-se os testes realizados relativamente à transferência de ficheiros e comentários acerca dos mesmos.

Relativamente à transferência do ficheiro penguin.gif exigida no guião, esta é realizada pela aplicação com distinto sucesso. De seguida estão apresentados prints da realização do referido teste.

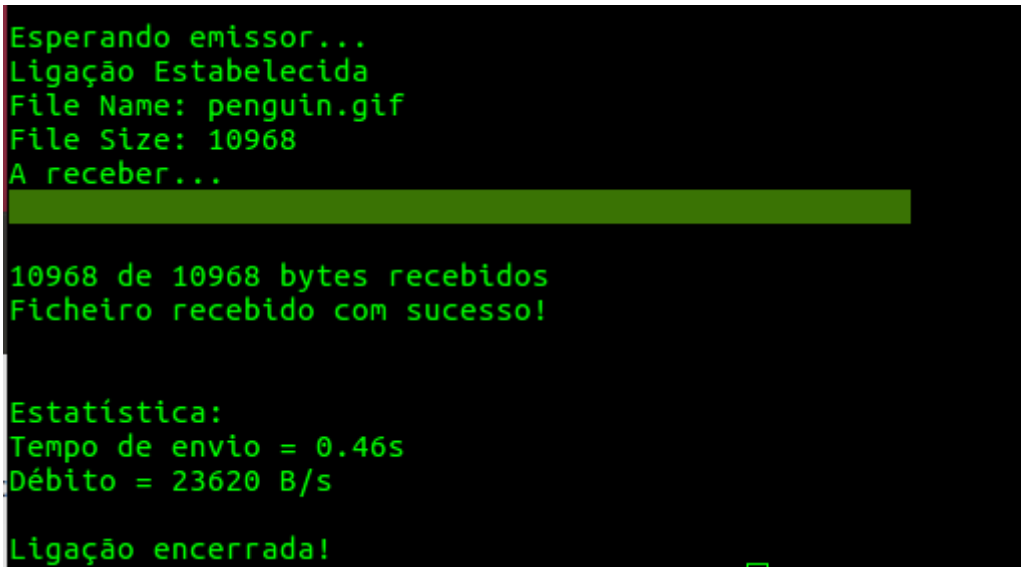

Fig. 1 - Modo Recetor

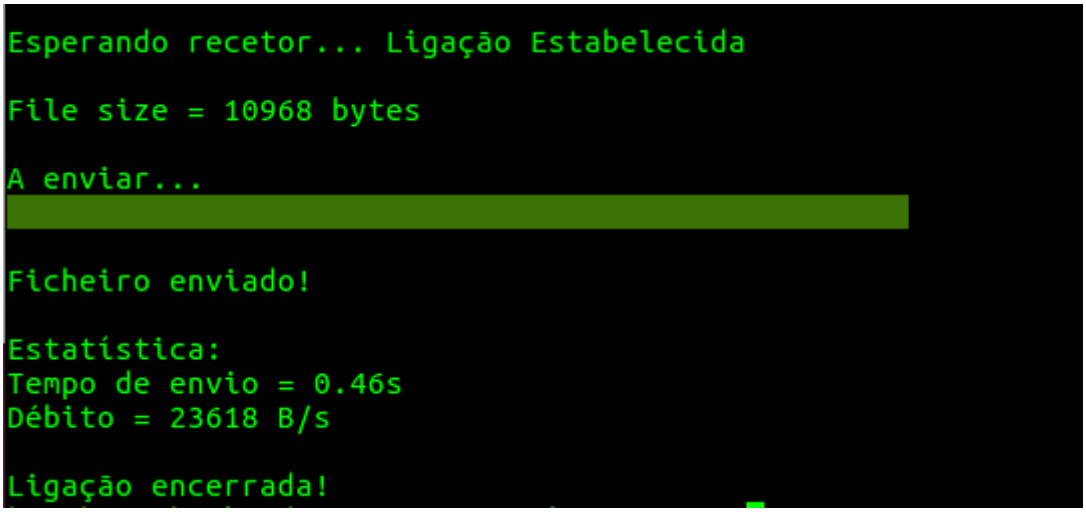

Fig. 2 - Modo Emissor

Além disso, também foi testado um ficheiro .gif de maior tamanho com sucesso idêntico ao anterior, como comprovado em seguida.

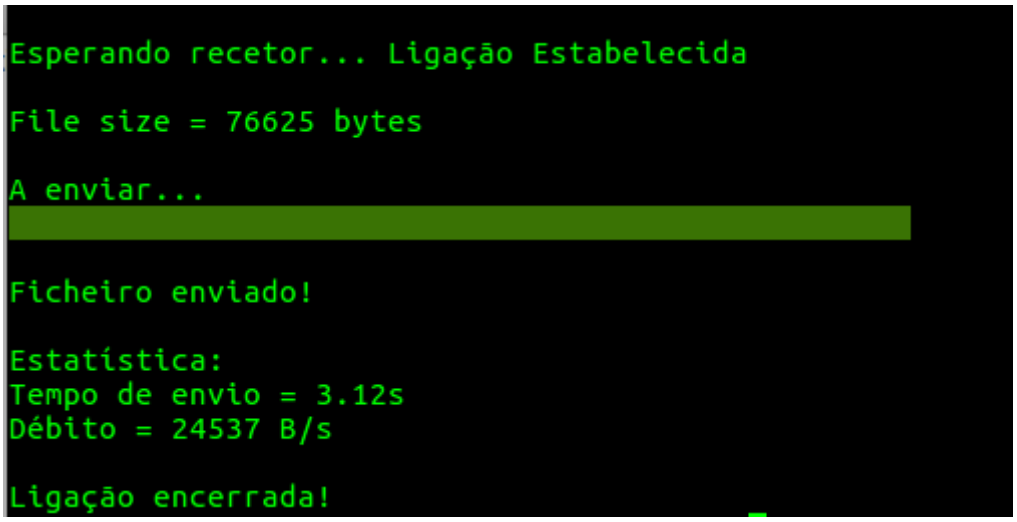

Fig. 3 - Modo Emissor com um ficheiro maior

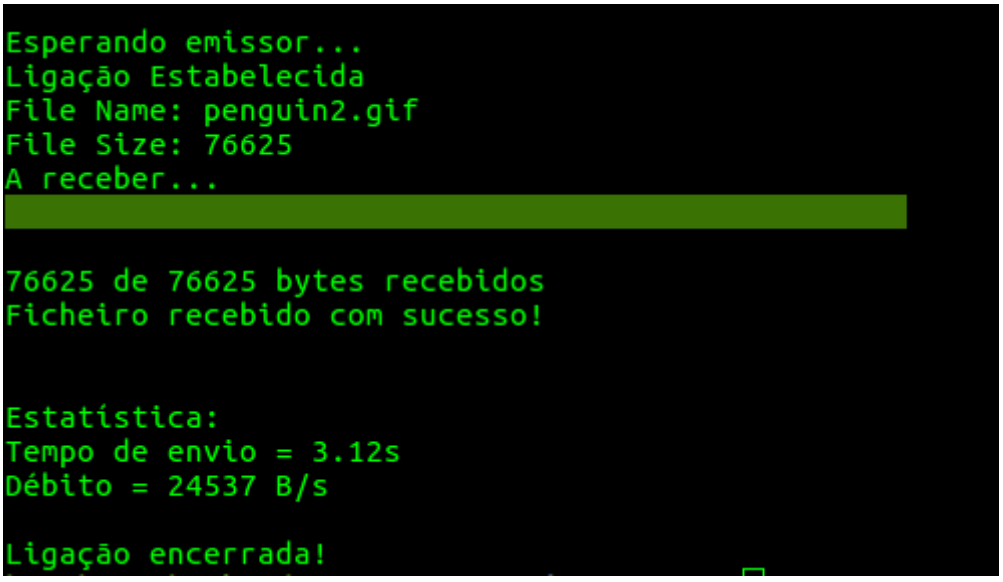

Fig. 4 - Modo Recetor com um ficheiro maior

Por fim, foram realizados testes onde a ligação da porta série foi cortada, de modo a testar os *timeouts* e se a ligação continuava após o restabelecimento da conexão; tais testes foram verificados com sucesso. Também foi testada uma a ligação à massa de pinos do lado do emissor de modo a introduzir erros, o dito teste foi igualmente foi passado com sucesso.

Apesar de não serem apresentadas imagens, estes testes foram comprovados pelo docente na altura da apresentação do projeto.

Elementos de valorização

Seleção do ficheiro a enviar

O utilizador pode selecionar qualquer ficheiro que pretenda enviar bastando, para isso, possuir o mesmo na pasta do projeto e indicar o seu nome no terminal.

Barra de progresso de envio/receção

Apresentação de uma barra de progresso no envio/receção do ficheiro.

Processamento das Tramas

É feito o processamento das tramas, sendo que:

- Tramas são processadas através das funções *control_frame()* e *header()*;
- Tramas com erros são verificadas através das variáveis de controlo REJ, RR, BCC e BCC2 e não são guardadas;

Verificação da Integridade dos Dados

A aplicação verifica se o tamanho do ficheiro recebido é igual ao tamanho do ficheiro enviado em *appLayer.c*

```
if(getFileSize(file) == received_file_size){

   printf("ld de %d bytes recebidos\n",getFileSize(file),received_file_size);
   printf("Ficheiro recebido com sucesso!\n\n");

}
else{
   printf("%ld de %d bytes recebidos\n",getFileSize(file),received_file_size);
   printf("Tamanho recebido diferente de tamanho original!\n\n");
}
```

Conclusões

Durante as últimas semanas quer através das aulas laboratoriais, quer através de tempo não letivo, o grupo desenvolveu esta aplicação que permite a transferência de dados entre dois computadores através de uma ligação física à porta série.

O trabalho foi bem entendido pelos elementos do grupo quer através da ajuda do docente da aula, que nos esclareceu diversos pontos acerca do mesmo, quer através do guião que nos serviu de guia.

Acerca da implementação do trabalho, os pontos pedidos no guião foram implementados, desde a divisão em camadas, o tratamento dos dados enviados através de tramas de controlo e o mecanismo de transparência de *byte stuffing*.

Em nota de conclusão, achamos que este trabalho contribuiu para a consolidação dos nossos conhecimentos acerca dos temas lecionados nas aulas teóricas e laboratoriais, permitindo-nos estar mais preparados para desafios futuros relacionados com este tema.

Anexo I - Código fonte

appLayer.c

```
1.  /*Non-Canonical Input Processing*/
2.
3.  #define CHUNK_SIZE 256                                          //Número d
    e bytes do ficheiro a ser enviado de cada vez
4.  #include "dataLink.h"
5.  #include "utilities.h"
6.
7.  int main(int argc, char** argv) {
8.
9.      fflush(NULL);
10.     int fd = open(argv[1], O_RDWR | O_NOCTTY );
11.
12.     struct termios oldtio,newtio;
13.
14.
15.     if (fd <0) {
16.         perror(argv[1]);
17.         exit(-1);
18.     }
19.
20.     if ( tcgetattr(fd,&oldtio) == -1) { /* save current port settings */
21.         perror("tcgetattr");
22.         exit(-1);
23.     }
24.
25.     bzero(&newtio, sizeof(newtio));
26.     newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
27.     newtio.c_iflag = IGNPAR;
28.     newtio.c_oflag = 0;
29.
30.     /* set input mode (non-canonical, no echo,...) */
31.     newtio.c_lflag = 0;
32.
33.     newtio.c_cc[VTIME]    = 1;   /* inter-character timer unused */
34.     newtio.c_cc[VMIN]     = 0;   /* blocking read until 5 chars received */
35.
36.
37.
38.     tcflush(fd, TCIOFLUSH);
39.
40.     if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
41.         perror("tcsetattr");
42.         exit(-1);
43.     }
44.
45.
46.     fflush(NULL);
47.
48.
49. /*-------------------------------------------------------------
50. -------------------------------------------------------------
51. -------------------------------------------------------------
52. ----------------Início da Comunicação-------------------------
53. -------------------------------------------------------------
54. -------------------------------------------------------------
55. -------------------------------------------------------------
56. */
57.     srand(time(NULL));
58.
```

```
59.
60.     //Tipo de comunicação: Sender (1) or Receiver (0)
61.     int com_type = check_arguments(argc, argv);
62.     struct timeval start, stop;
63.     double secs = 0;
64.     FILE *file;
65.
66.     printf("\n");
67.
68.     if(com_type) {
69.
70.
71.         if((file=openfile(argv[3], com_type))==NULL)exit(1);
72.         if(LLOPEN(fd , com_type)==-
    1) exit(1);                                          //Estabelecimento da comunica
    ção
73.         gettimeofday(&start, NULL);
          //Inicia relógio
74.         fflush(NULL);
75.
76.
77.
78.         char buffer[CHUNK_SIZE];
79.         char *payload;
80.         char *stuffed;
81.         int size;
82.         short sequence_number = 0;
83.         char *control = control_frame( argv[3] , file , 1 , &size);
          //START frame
84.
85.         stuffed = stuffing (control, &size);

86.
87.
88.         if(LLWRITE(fd, stuffed, size) == -
    1 )                                          //Send file info
89.             exit(1);
90.
91.         printf("A enviar...\n");
92.
93.         int enviado=0;
94.         int file_size = getFileSize(file);
95.
96.         while ( (size = fread(buffer, sizeof(char), CHUNK_SIZE, file)) > 0){
       //Lê fragmento do ficheiro
97.                 enviado = enviado + size;
98.                 loading(enviado,file_size);
          //Barra de progresso
99.
100.
101.                        payload = header(buffer, &size, sequence_number++);
                 //Adiciona campo de controlo, número de sequência, tamanho da pay
    load
102.                        stuffed = stuffing(payload, &size);
                 //Transparência e adiciona BCC2
103.
104.                        if(LLWRITE(fd, stuffed, size)==-
    1)                                   //Envia o trama
105.                            exit(1);
106.
107.              }
108.
109.              control = control_frame( argv[3] , file , 0 , &size);
                 //END frame
110.              stuffed = stuffing (control, &size);
111.
```

```
112.                if(LLWRITE(fd, stuffed, size) == -
    1 )                                     //Send END frame
113.                    exit(1);
114.
115.                gettimeofday(&stop, NULL);
                  //Para o relógio
116.                printf("\n\nFicheiro enviado!\n");
117.                secs = (double)(stop.tv_usec -
     start.tv_usec) / 1000000 + (double)(stop.tv_sec - start.tv_sec);
118.                printf("\nEstatística:\n");
119.                printf("Tempo de envio = %.2fs\n",secs);
120.                printf("Débito = %.0f B/s\n\n", file_size/secs);
121.
122.            }
123.
124.            if(!com_type){
125.
126.                if(LLOPEN(fd , com_type)==-
    1) exit(1);                                          //Estabelecimento da comunica
    ção
127.                gettimeofday(&start, NULL);
                  //Inicia o relógio
128.
129.                char stuffed [CHUNK_SIZE+(int)CHUNK_SIZE/2];

130.                char *payload;
131.                char *buffer;
132.                int length;
133.                int received_file_size=0;
134.                char *filename;
135.
136.                //Leitura da informação do ficheiro
137.                while(1){
138.                    length = LLREAD(fd, stuffed);
139.
140.                    buffer = verify_bcc2(stuffed, &length);
141.                    if(buffer == NULL)
142.                            send_REJ(fd);
143.                    else{
144.                        filename = get_info(buffer, &received_file_size);
145.                        if(filename!=NULL){
146.                            printf("File Name: %s\nFile Size: %d\n", filename,
    received_file_size);
147.                            send_RR(fd);
148.                            file=openfile(filename, com_type);
149.                            break;
150.                        }
151.                        else
152.                            send_REJ(fd);
153.                    }
154.                }
155.
156.                //Receção do ficheiro
157.                printf("A receber...\n");
158.                while( (length = LLREAD(fd, stuffed) )> 0){
159.
160.                    loading((int)getFileSize(file),received_file_size);
161.                    if(stuffed[0]==END){
162.
163.                        buffer = verify_bcc2(stuffed, &length);
164.                        if(buffer == NULL)
165.                            send_REJ(fd);
166.                        else{
167.                            send_RR(fd);
168.                            break;
169.                        }
```

```
170.                    }
171.                    else{
172.                        payload = verify_bcc2(stuffed, &length);
                    //Faz destuff e verifica o BCC2
173.
174.                        if(payload == NULL)
175.                                send_REJ(fd);
176.                        else{
177.                            buffer = remove_header(payload, &length);
                    //Remove Header
178.                            send_RR(fd);
179.                            fwrite(buffer,1,length,file);
180.                        }
181.
182.                    }
183.
184.                }
185.
186.                gettimeofday(&stop, NULL);
187.
188.                if(getFileSize(file) == received_file_size)
189.                    printf("\n\n%ld de %d bytes recebidos\nFicheiro recebido co
    m sucesso!\n\n", getFileSize(file),received_file_size);
190.                else
191.                    printf("\n\n%ld de %d bytes recebidos\nTamanho recebido dif
    erente de tamanho original!\n\n", getFileSize(file),received_file_size);
192.                secs = (double)(stop.tv_usec -
     start.tv_usec) / 1000000 + (double)(stop.tv_sec - start.tv_sec);
193.                printf("\nEstatística:\n");
194.                printf("Tempo de envio = %.2fs\n",secs);
195.                int file_size = getFileSize(file);
196.                printf("Débito = %.0f B/s\n\n", received_file_size/secs);
197.                fclose(file);
198.
199.            }
200.
201.
202.            LLCLOSE(fd, com_type);
203.
204.
205.
206.
207.
208.
209.
210.            if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
211.                perror("tcsetattr");
212.                exit(-1);
213.            }
214.
215.
216.
217.
218.            close(fd);
219.            return 0;
220.        }
```

dataLink.c

```
#include "dataLink.h"

int flag=0;
int timeout=0;
int Nr=0;


void time_out() {
    timeout++;
    flag=1;
}

int LLOPEN(int fd, int com_type) {

    if(com_type)
    (void) signal(SIGALRM, time_out);
    char address, address2;
    char c;
    int state = 0;

    if(!com_type) {
        printf("Esperando emissor...\n");
    }
    else
        printf("Esperando recetor...\n");
    while(timeout<=TIMEOUTS) {

        if(com_type) {

            send_SET(fd);
            alarm(3);
            flag=0;
        }


        while(state != 5 && flag==0 ) {

            read(fd, &c, 1);


            switch (state) {

            case 0:                                     //FLAG
                if(c == FLAG) {
                    state = 1;
                }//else stay in same state
                break;
            case 1:                                     //A_R/A_T
                if(com_type) {
                    address=A_R;
                }
                else {
                    address=A_T;
                }
                if(c == address) {
                    state = 2;
                } else if(c == FLAG) {
                    state = 1;

                } else
                    state=0;
                break;
```

```
64.
65.             case 2:                                     //UA/SET
66.                 if(com_type) {
67.                     address2=UA;
68.                 }
69.                 else {
70.                     address2=SET;
71.                 }
72.                 if(c == address2) {
73.                     state = 3;
74.                 } else if(c == FLAG) {
75.                     state = 1;
76.                 } else {
77.                     state = 0;
78.                 }
79.                 break;
80.             case 3:                                     //BCC
81.                 if (c == (address^address2)) {
82.                     state = 4;
83.                 } else {
84.                     state = 0;
85.                 }
86.                 break;
87.             case 4:                                     //FLAG
88.                 if (c == FLAG) {
89.                     state = 5;
90.                     if (!com_type) {
91.                         send_UA(fd, com_type);
92.                     }
93.                     printf("Ligação Estabelecida\n");
94.                     return 1;
95.                 } else {
96.                     state = 0;
97.                 }
98.                 break;
99.             }
100.            }
101.        }
102.
103.        printf("timeout - Ligação Não Estabelecida\n");
104.        return -1;
105.    }
106.
107.    int LLWRITE(int fd, char *buffer, int length) {
108.
109.        fflush(NULL);
110.        timeout = 0;
111.        char *trama = malloc((length+5)*sizeof(char));
112.        char controlo;
113.        char c;
114.        int i, written, state = 0;
115.
116.        if(Nr==0) {
117.            Nr=1;
118.            controlo = RR0;
119.        }
120.        else if(Nr==1) {
121.            Nr=0;
122.            controlo = RR1;
123.        }
124.
125.
126.        //Prepara bytes iniciais
127.        trama[0] = FLAG;
128.        trama[1] = A_T;
129.        trama[2] = controlo;
```

```
130.            trama[3] = trama[1]^trama[2];
131.            trama[length+4] = FLAG ;
132.
133.            for(i = 4 ; i < length + 4 ; i++) {
134.                trama[i]=buffer[i-4];
135.            }
136.
137.
138.
139.            //ESPERAR PELO ACK
140.            (void) signal(SIGALRM, time_out);
141.            timeout = 0;
142.            while(timeout<=TIMEOUTS) {
143.
144.                written = write(fd, trama, length + 5);
145.                written = written-5;
146.
147.
148.                alarm(3);
149.                flag=0;
150.                state=0;
151.                while(state != 5 && flag==0 ) {
152.
153.
154.                    read(fd, &c, 1);
155.
156.
157.                    switch (state) {
158.                    case 0:                                         //FLAG
159.                        if(c == FLAG) {
160.                            state = 1;
161.                        }//else stay in same state
162.                        break;
163.                    case 1:                                         //A_T/A_R
164.
165.                        if(c == A_T) {
166.                            state = 2;
167.                        } else if(c == FLAG) {
168.                            state = 1;
169.
170.                        } else
171.                            state=0;//else go back to beggining
172.                        break;
173.
174.                    case 2:                                         //RR/REJ
175.
176.                        if(c == RR0 || c == RR1) {
177.
178.                            state = 3;
179.                        }
180.                        else if( c == REJ0 || c == REJ1){
181.
182.                            state = 0;
183.                            flag = 1;
184.
185.                        } else if(c == FLAG) {                      //FLAG
186.                            state = 1;
187.                        } else {
188.                            state = 0;
189.                        }
190.                        break;
191.                    case 3:                                         //BCC
192.                        if (c == (A_T^controlo)) {
193.                            state = 4;
194.                        } else {
195.                            state = 0;
```

```
196.                        }
197.                        break;
198.                    case 4:                                         //FLAG
199.                        if (c == FLAG) {
200.                            state = 5;
201.                            return written;
202.
203.                        } else {
204.                            state = 0;
205.                        }
206.                        break;
207.                    }
208.                }
209.            }
210.            free(trama);
211.            printf("TIMEOUT - Escrita não Realizada\n");
212.            return -1;
213.
214.        }
215.
216.        int LLREAD(int fd, char *buffer) {
217.
218.            int length=0;
219.            int random;
220.            int state = 0;
221.            char c;
222.            char controlo;
223.
224.           if(Nr==0) {
225.
226.                controlo = RR0;
227.
228.
229.            }
230.            else if(Nr==1) {
231.
232.                controlo = RR1;
233.            }
234.
235.
236.                while(state != 5) {
237.
238.                        read(fd, &c, 1);
239.
240.                       switch (state) {
241.                        case 0:                                     //FLAG
242.                            if(c == FLAG) {
243.                                state = 1;
244.                            }
245.                            break;
246.                        case 1:                                     //A
247.
248.                            if(c == A_T) {
249.                                state = 2;
250.                            } else if(c == FLAG) {
251.                                state = 1;
252.
253.                            } else
254.                                state=0;
255.                            break;
256.
257.                        case 2:                                     //RR
258.
259.                            if(c == controlo) {
260.                                state = 3;
261.                            } else if(c == FLAG) {
```

```
                                state = 1;
                            } else {
                                state = 0;
                            }
                            break;
                        case 3:                                     //BCC

                            if(BCC_ERROR_PROBABILITY != 0){     //Introdução ma
    nual de erro de leitura
                                random = rand();
                                random = (random % (100/BCC_ERROR_PROBABILITY))
     + 1;

                                if(random == 1)
                                    c = !c;
                            }

                            if (c == ((A_T^controlo))) {
                                state = 4;
                            } else
                                state = 0;//else go back to beggining

                            break;
                        case 4:                                     //FLAG
                            if( c == FLAG){
                                state = 5;
                            }
                            else{
                                buffer[length] = c;
                                length++;
                            }


                            break;
                        }
                    }
        if(TRANSMISSION_DELAY)
            delay(TRANSMISSION_DELAY*2);
        return length;



        }

        int LLCLOSE(int fd, int com_type) {

            char address;
            int state = 0;
            char c;
            int receiveUA = 0;


            if(com_type)
                send_DISC(fd, com_type);



            while(timeout<=TIMEOUTS || !com_type) {

                if(com_type) {
                    alarm(3);
                    flag=0;
                }

```

```
326.                while(state != 5 && flag==0 ) {
327.
328.                    read(fd, &c, 1);
329.
330.                    switch (state) {
331.                    case 0:                                     //FLAG
332.                        if(c == FLAG) {
333.                            state = 1;
334.                        }
335.                        break;
336.                    case 1:                                     //A
337.                        if(com_type) {
338.                            address=A_R;
339.                        }
340.                        else if(!com_type){
341.                            address=A_T;
342.                        }
343.                        if(c == address) {
344.                            state = 2;
345.                        }else if(c==FLAG){
346.                            state = 1;
347.
348.                        } else
349.                            state=0;
350.                        break;
351.
352.                    case 2:
353.
354.                        if(c == DISC) {                         //DISC
355.                            state = 3;
356.                        } else if(c == FLAG) {
357.                            state = 1;
358.                        }
359.                        else if(c == UA && receiveUA == 1){
360.                            state = 3;
361.                        } else {
362.                            state = 0;
363.                        }
364.                        break;
365.                    case 3:                                     //BCC
366.
367.                        if (c == (address^DISC)) {
368.                            state = 4;
369.                        }
370.                        else if((receiveUA == 1) && (c == (UA^address))){
371.
372.                            state = 4;
373.                        }
374.                        else {
375.                            state = 0;
376.                        }
377.
378.                        break;
379.                    case 4:                                     //FLAG
380.
381.                        if (c == FLAG) {
382.                            state = 5;
383.                            if (!com_type && receiveUA == 0) {
384.                                send_DISC(fd, com_type);
385.
386.                                state = 0;
387.                                receiveUA = 1;
388.                            }
389.                            else if(com_type){
390.                                send_UA(fd, com_type);
391.
```

```
392.                                 printf("Ligação encerrada!\n");
393.                             return 1;
394.                             }
395.                             else if(receiveUA == 1){
396.
397.                                 printf("Ligação encerrada!\n");
398.                                 return 1;
399.                             }
400.
401.                         } else {
402.                             return -1;
403.                         }
404.                         break;
405.                     }
406.                 }
407.             }
408.
409.
410.
411.             return -1;
412.         }
413.
414.         void send_UA(int fd, int com_type) {
415.
416.             char trama_UA[5];
417.             char address;
418.             trama_UA[0]=FLAG;
419.             if(com_type){
420.                 address = A_T;
421.                 trama_UA[1]=address;
422.             }
423.             else{
424.                 address = A_R;
425.                 trama_UA[1] = address;
426.             }
427.             trama_UA[2]=UA;
428.             trama_UA[3]=address^UA;
429.             trama_UA[4]=FLAG;
430.
431.             write(fd, trama_UA, 5);
432.             fflush(NULL);
433.
434.         }
435.
436.         void send_SET(int fd) {
437.
438.             char trama[5];
439.             trama[0]=FLAG;
440.             trama[1]=A_T;
441.             trama[2]=SET;
442.             trama[3]=A_T^SET;
443.             trama[4]=FLAG;
444.
445.             write(fd, trama, 5);
446.             fflush(NULL);
447.
448.         }
449.
450.         void send_DISC(int fd, int com_type) {
451.
452.             char trama[5];
453.             char address;
454.             trama[0]=FLAG;
455.             if(com_type){
456.                 address = A_T;
457.                 trama[1] = address;
```

```
458.            }
459.            else{
460.                address = A_R;
461.                trama[1] = address;
462.            }
463.            trama[2]=DISC;
464.            trama[3]=address^DISC;
465.            trama[4]=FLAG;
466.
467.            write(fd, trama, 5);
468.            fflush(NULL);
469.
470.        }
471.
472.        void send_RR(int fd){
473.
474.            char controlo;
475.            if(Nr==0) {
476.
477.                Nr = 1;
478.                controlo = RR0;
479.            }
480.            else if(Nr==1) {
481.
482.                Nr = 0;
483.                controlo = RR1;
484.            }
485.
486.            char trama[5];
487.            trama[0]=FLAG;
488.            trama[1]=A_T;
489.            trama[2]=controlo;
490.            trama[3]=(A_T^controlo);
491.            trama[4]=FLAG;
492.
493.            write(fd, trama, 5);
494.            fflush(NULL);
495.
496.
497.        }
498.        void send_REJ(int fd){
499.            char controlo;
500.            if(Nr==0) {
501.
502.                controlo = REJ0;
503.            }
504.            else if(Nr==1) {
505.
506.                controlo = REJ1;
507.            }
508.
509.            char trama[5];
510.            trama[0]=FLAG;
511.            trama[1]=A_T;
512.            trama[2]=controlo;
513.            trama[3]=(A_T^controlo);
514.            trama[4]=FLAG;
515.
516.            write(fd, trama, 5);
517.            fflush(NULL);
518.
519.        }
520.
521.        char* verify_bcc2(char* control_message, int* length){
522.
523.            char* destuffed_message = destuffing(control_message, length);
```

```
524.
525.            int i;
526.            int random;
527.            char control_bcc2 = 0x00;
528.            for(i=0; i<*length-1; i++){
529.                control_bcc2 ^= destuffed_message[i];
530.
531.            }
532.            if(BCC2_ERROR_PROBABILITY != 0){
533.                                    random = rand();
534.                                    random = (random % (100/BCC2_ERROR_PROBABILITY)
    ) + 1;
535.
536.                                    if(random == 1)
537.                                        control_bcc2 = !control_bcc2;
538.                                }
539.            if(control_bcc2 != destuffed_message[*length-1]){
540.                *length = -1;
541.                return NULL;
542.            }
543.            *length = *length-1;
544.            char* data_message = (char*) malloc(*length);
545.            for(i=0; i<*length; i++){
546.                    data_message[i] = destuffed_message[i];
547.            }
548.            free(destuffed_message);
549.
550.
551.            return data_message;
552.        }
553.
554.        char* destuffing(char* msg, int* length){
555.
556.
557.            char* str = (char*) malloc(*length);
558.            int i;
559.            int new_length = 0;
560.
561.
562.
563.            for(i=0; i<*length; i++){
564.                new_length++;
565.
566.                if(msg[i] == 0x7d){
567.                    if(msg[i+1] == 0x5e){
568.                        str[new_length-1] = FLAG;
569.                        i++;
570.                    }
571.                    else if(msg[i+1] == 0x5d){
572.                        str[new_length -1] = 0x7d;
573.                        i++;
574.                    }
575.                }
576.                else{
577.                    str[new_length-1] = msg[i];
578.                }
579.
580.            }
581.            *length = new_length;
582.
583.            return str;
584.        }
585.
586.        char* stuffing(char* payload, int* length){
587.
588.            char* str;
```

```
589.            char* msg_aux;
590.            int i;
591.            int j;
592.
593.
594.            str = (char *) malloc(*length);
595.            msg_aux = (char *) malloc(*length);
596.            char BCC2=0x00;
597.
598.            for(i=0,j=0; i < *length; i++, j++){
599.                BCC2^=payload[i];
600.                msg_aux[i] = payload[i];
601.            }
602.
603.
604.            msg_aux = (char*) realloc(msg_aux,(*length)+1);
605.            msg_aux[*length/*index = length-1*/]=BCC2;
606.
607.            for(i=0,j=0; i < *length+1; i++, j++){
608.
609.
610.                if(msg_aux[i] ==  FLAG){
611.                    str[j] = 0x7d;
612.                    str[j+1] = 0x5e;
613.                    j++;
614.                }
615.                else if(msg_aux[i] == 0x7d){
616.                    str[j] = 0x7d;
617.                    str[j+1]= 0x5d;
618.                    j++;
619.                }
620.                else{
621.                    str[j] = msg_aux[i];
622.                }
623.            }
624.
625.
626.
627.
628.            *length=j;
629.
630.            return str;
631.        }
632.
633.        char* control_frame(char* filename, FILE *file, int start, int* frame_s
    ize){
634.
635.
636.            int file_name_size  = strlen(filename);
637.            int file_size = getFileSize(file);                          //Get f
    ile size
638.            if(start)
639.                printf("\nFile size = %d bytes\n\n",file_size);
640.            int i = 0;
641.            char file_size_in_string[30];
642.            sprintf(file_size_in_string, "%d", file_size);
643.
644.            *frame_size = 5 + file_name_size + strlen(file_size_in_string);
645.            char *control_frame = malloc(*frame_size);
646.
647.            if(start)
648.                control_frame[i++] = START;
649.            else
650.                control_frame[i++] = END;
651.
652.
```

```
653.            control_frame[i++] = 0x00;
654.            control_frame[i++] = (char)strlen(file_size_in_string);
655.
656.
657.            for(; i < strlen(file_size_in_string)+3 ; i++){
658.
659.                control_frame[i] = file_size_in_string[i-3];
660.            }
661.
662.
663.            control_frame[i++] = 0x01;
664.            control_frame[i++] = (char) file_name_size;
665.
666.            int j;
667.            for( j=i ; i<file_name_size+j ; i++ ){
668.
669.                control_frame[i] = filename[i-j];
670.            }
671.
672.            return control_frame;
673.        }
674.
675.        char* get_info(char* control, int* file_size){
676.
677.            if(control[0]!=0x01)return NULL;
678.            int pos = 4+control[2];
679.            int filename_size = control[4+control[2]];
680.
681.            char *buffer = malloc(100);
682.
683.            char* size = malloc(control[2]);
684.            int i;
685.
686.            for(i=0 ; i < filename_size ; i++ ){
687.                buffer[i] = control[pos+1+i];
688.            }
689.
690.
691.            for(i=0 ; i < control[2] ; i++ )
692.                size[i] = control[i+3];
693.
694.
695.            *file_size = atoi(size);
696.            return buffer;
697.        }
698.
699.        char* header(char* buffer, int* length, short sequence_number){
700.
701.            char* str = malloc((*length)+4);
702.
703.            str[0] = 0x00;
704.            str[1] = (char)sequence_number;
705.            str[2] = (char)(*length)/256;
706.            str[3] = (char)(*length)%256;
707.
708.            int i;
709.            for(i = 0 ; i < *length ; i++ ){
710.                str[i+4] = buffer[i];
711.            }
712.
713.            *length = *length + 4;
714.            return str;
715.
716.        }
717.
718.        char* remove_header(char* buffer, int* length){
```

```
719.
720.            char* str = malloc(2 * (*length));
721.            int i;
722.
723.
724.
725.            for(i = 0 ; i < *length - 4 ; i++)
726.                str[i] = buffer[i+4];
727.
728.            *length = *length-4;
729.            return str;
730.
731.
732.        }
```

dataLink.h

```
1.  #include <sys/types.h>
2.  #include <sys/stat.h>
3.  #include <fcntl.h>
4.  #include <termios.h>
5.  #include <stdio.h>
6.  #include <signal.h>
7.  #include <unistd.h>
8.  #include <stdbool.h>
9.  #include <stdlib.h>
10. #include <string.h>
11. #include <sys/time.h>
12. #include "utilities.h"
13.
14. //-------------------------------------------------------------------------
    ------------------------------------------------------
15. #define BCC_ERROR_PROBABILITY   1       //Probabilidade de erro na leitura de
    BCC ----------Valores entre 0-50  ------------------------
16. #define BCC2_ERROR_PROBABILITY  0       //Probabilidade de erro na leitura de
    BCC2  --------Valores entre 0-50 -------------------------
17. #define TRANSMISSION_DELAY      5       //Tempo simulado de propagação dos pac
    otes (ms) ----------------------------------------------
18. #define TIMEOUTS                3       //Número de timeouts de 3s até encerra
    r a ligação
19. //-------------------------------------------------------------------------
    ------------------------------------------------------
20.
21. #define TRANSMITTER 1
22. #define RECEIVER 0
23.
24.
25. #define BAUDRATE B38400
26. #define MODEMDEVICE "/dev/ttyS1"
27. #define _POSIX_SOURCE 1 /* POSIX compliant source */
28. #define FALSE 0
29. #define TRUE 1
30.
31. #define OPEN 0
32. #define READ 1
33. #define WRITE 2
34.
35. #define FLAG 0x7E
36. #define C_T 0x03
37. #define C_R 0x07
38. #define A_T 0x03
```

```
39. #define A_R 0x01
40. #define BCC_T 0x00
41. #define BCC_R 0x04
42.
43. #define UA 0x07
44. #define SET 0x03
45. #define DISC 0x0A
46. #define START 0x01
47. #define END 0x02
48.
49. #define RR0 0x05
50. #define RR1 0x25
51. #define REJ0 0x01
52. #define REJ1 0x21
53.
54.
55. #define TRAMA_S 0
56. #define TRAMA_I 1
57.
58. #define ERROR_BCC1 1
59. #define ERROR_BCC2 2
60.
61. void time_out();
62.
63.
64.
65. //-------------------------------------------------Data Link Layer -----------
    ---------------
66.
67. int LLWRITE(int fd, char *buffer, int length);
      //Envia o trama. Retorna nr de chars escritos. -1 em caso de erro
68.
69. int LLOPEN(int fd, int com_type);
      //Estabelece a comunicação. Retorna 1 ou -1(erro)
70.
71. int LLREAD(int fd, char *buffer);
      //Retorna número de chars de Data lidos
72.
73. int LLCLOSE(int fd, int com_type);
      //Encerra a comunicação. Retorna 1 ou -1 (erro)
74.
75. //Funções de stuffing, destuffing e a função que verifica o BCC2
76. char* stuffing(char* payload, int* length);
      //Faz stuffing e adiciona BCC2
77. char* destuffing(char* msg, int* length);
      //Faz destuffing e retira BCC2
78. char* verify_bcc2(char* control_message, int* length);
      //Verifica BCC2
79. char* control_frame(char* filename, FILE *file, int start, int* frame_size);
      //Cria pacote de controlo START/END
80. char* get_info(char* control, int* file_size);
      //Extrai nome de ficheiro e respetivo tamanho em bytes
81. char* header(char* buffer, int* length, short sequence_number);
      //Adiciona byte de controlo, número de sequência e tamanho do campo de dados

82. char* remove_header(char* buffer, int* length);
      //Remove Header
83.
84. FILE *openfile(char* filename, int com_type);
85. void send_SET(int fd);
86. void send_UA(int fd, int com_type);
87. void send_DISC(int fd, int com_type);
88. void send_REJ(int fd);
89. void send_RR(int fd);
```

utilities.c

```
1.  #include "utilities.h"
2.
3.  FILE *openfile(char* filename, int com_type){
4.
5.      FILE *file;
6.
7.      if(com_type) file=fopen(filename, "rb");
8.
9.
10.     else file=fopen(filename, "wb");
11.
12.     if(file==NULL){
13.         printf("Erro a abrir o ficheiro %s\n", filename);
14.         return NULL;
15.     }
16.
17.     return file;
18. }
19.
20. void delay(int milliseconds){
21.     long pause;
22.     clock_t now,then;
23.
24.     pause = milliseconds*(CLOCKS_PER_SEC/1000);
25.     now = then = clock();
26.     while( (now-then) < pause )
27.         now = clock();
28. }
29.
30. long getFileSize(FILE* file) {
31.
32.     long currentPosition = ftell(file);
33.
34.
35.     if (fseek(file, 0, SEEK_END) == -1) {
36.         printf("ERROR: Could not get file size.\n");
37.         return -1;
38.     }
39.
40.
41.     long size = ftell(file);
42.
43.
44.     fseek(file, 0, currentPosition);
45.
46.
47.     return size;
48. }
49.
50. int check_arguments(int argc, char** argv){
51.
52.     int com_type;
53.     if ( (argc < 2) ||
54.             (
55.             (strcmp("/dev/ttyS0", argv[1])!=0) && (strcmp("/dev/ttyS1", argv[1
    ])!=0) )) {
56.         printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1\n");
57.         exit(1);
58.     }
59.
```

```
60.     if((argc < 3) ||
61.                ( (strcmp("sender", argv[2])!=0) && (strcmp("receiver", argv[2])!
    =0))){
62.                    printf("Specify if 'sender' or 'receiver' and file name\n");
63.                    exit(1);
64.                }
65.     if(argc < 4 && strcmp(argv[2],"sender") == 0){
66.         printf("Specify file name\n");
67.         exit(1);
68.     }
69.     else{
70.         if(strcmp("sender", argv[2])==0){
71.             com_type = 1;
72.         }
73.         else if(strcmp("receiver", argv[2])==0){
74.             com_type = 0;
75.         }
76.         else{
77.             printf("Specify if 'sender' or 'receiver'\n");
78.         }
79.     }
80.
81.     return com_type;
82. }
83.
84. void loading(int current_size, int total_size){
85.
86.     static short progress=0;
87.     int i;
88.     if(progress == 0){
89.         for(i=0 ; i < 50 ; i++)  printf("\x1B[33m\u2591\x1B[0m");
               //IMPRIME BACKGROUND
90.         for(i=0 ; i < 50 ; i++)  printf("\b");
               //VOLTA AO INICIO DA LINHA
91.     }
92.
93.
94.     for(i = 0 ; i < ((int)((float)current_size/(float)total_size*50)) -
     progress ; i++)
95.         printf("\x1B[32m\u2593\x1B[0m");
                 //IMPRIME BARRA
96.
97.     progress=(int)((float)current_size/(float)total_size*50);
98. }
```

utilities.h

```
1.  #include <time.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <string.h>
5.
6.
7.  FILE *openfile(char* filename, int com_type);           //Lê ou cria um fichei
    ro
8.  void delay(int milliseconds);                          //Self-explanatory
9.  long getFileSize(FILE* file);                          //Retorna o tamanho do
     ficheiro
10. int check_arguments(int argc, char** argv);            //Verifica se os argum
    entos estão certos e imprime informação em caso contrário. Devolve o tipo de c
    omunicação
11. void loading(int current_size, int total_size);        //Barra de progresso
```

Anexo II – Estatísticas realizadas

Tamanho do ficheiro (bytes)	Baudrate
10968	38400

Chunk Size	tempo (s)	R (bytes/s)	R (bit/s)	S (R/C)
20	22,16	495	3960	0,103
40	11,14	984,5	7876	0,205
60	7,45	1472,2	11777,6	0,307
80	5,64	1945,3	15562,4	0,405
100	4,51	2431,3	19450,4	0,507
120	3,79	2893,6	23148,8	0,603
140	3,27	3359,2	26873,6	0,700
160	2,86	3830,4	30643,2	0,798
180	2,55	4307,2	34457,6	0,897
200	2,31	4756,1	38048,8	0,991

Fig. 5 - Tabelas da eficiência variando o tamanho das tramas.

Tamanho do ficheiro (bytes)	Chunk Size
10968	100

Baudrate (C)	tempo (s)	R (bytes/s)	R (bit/s)	S (R/C)
600	220,06	49,84095247	398,7276197	0,665
1200	110,98	98,82861777	790,6289422	0,659
1800	71,3	153,828892	1230,631136	0,684
2400	57,09	192,1177089	1536,941671	0,640
4800	30,99	353,9206196	2831,364956	0,590
9600	14,03	781,7533856	6254,027085	0,651
19200	6,89	1591,872279	12734,97823	0,663
38400	4,51	2431,929047	19455,43237	0,507

Fig. 6 – Tabelas da eficiência variando o Baudrate.

Tamanho do ficheiro (bytes)	Baudrate	Chunk Size
10968	38400	100

prob erro(bcc1+bcc2)%	tempo (s)	R (bytes/s)	R (bit/s)	S (R/C)
0+0	4,51	2430	19440	0,506
2+0	4,62	2375,7	19005,6	0,495
0+2	4,55	2410,3	19282,4	0,502
2+2	11,9	997	7976	0,208
4+2	14,9	770,7	6165,6	0,161
2+4	12,96	858,3	6866,4	0,179
4+4	18	633,3	5066,4	0,132
6+4	37,78	299	2392	0,062
4+6	15,24	768,3	6146,4	0,160
6+6	29,57	373,7	2989,6	0,078
8+6	35,92	330,3	2642,4	0,069
6+8	25,63	483	3864	0,101
8+8	41,14	269,3	2154,4	0,056
10+8	43,1	255,7	2045,6	0,053
8+10	33,9	343,7	2749,6	0,072
10+10	46,3	245,7	1965,6	0,051

Fig. 7 – Tabelas da eficiência variando a % de erros