REDES DE COMPUTADORES

RELATÓRIO DO 1º TRABALHO LABORATORIAL

David Ferreira (up202006302)
Henrique Vicente (up202005321)

Sumário

Este relatório, realizado na unidade curricular Redes de Computadores, tem como objetivo descrever o primeiro trabalho laboratorial, que consistiu em desenvolver uma aplicação capaz de transferir ficheiros de um computador para o outro, através de uma porta de série.

Após a conclusão do trabalho, foi realizada a transferência de dados sem qualquer perda de dados ou ocorrência de erros. Nos casos em que existiu perda de ligação com a porta série, o programa foi capaz de restabelecer a transmissão e recuperar os dados.

Introdução

Com a realização deste trabalho, é pretendida, não só a implementação de um protocolo de ligação de dados com determinadas características, como, por exemplo, a transmissão estruturada em tramas e a transparência na transmissão de dados, como também testar o protocolo com uma aplicação de transferência de ficheiros e determinar a sua eficiência.

Relativamente à aplicação, são necessários dois tipos de pacotes de controlo que controlam a transmissão: o start e o end.

Neste relatório, explicaremos passo a passo as fases do protocolo em vários pontos:

- **Arquitetura:** Demonstração dos blocos funcionais e interfaces;
- **Estrutura do código:** Estruturas de dados utilizadas, funções e a sua relação com a arquitetura;
- **Casos de uso principais:** Identificação e descrição de chamada de funções;
- **Protocolo de ligação lógica:** Descrição da estratégia de implementação dos aspetos funcionais;
- **Protocolo de aplicação:** Identificação e descrição dos principais aspetos funcionais;
- **Validação:** Descrição dos testes efectuados com apresentação dos resultados obtidos;
- **Eficiência do protocolo de ligação de dados:** Estatística da eficiência do protocolo;
- **Conclusão:** Síntese do trabalho realizado e reflexão dos objetivos alcançados.

Arquitetura

O projeto é dividido por dois blocos funcionais,emissor e receptor, tendo cada um uma camada de ligação. Para tal, implementamos ficheiros source **(E_Application_layer.c**, **R_ Application_layer.c**, **E_main.c**, **R_main.c**) e headers (**E_ Application_layer.h**, **R_ Application_layer.h**),(**E_ link_layer.h**, **R_link_layer.h)**.

A camada de ligação de dados tem como função estabelecer a ligação. Aqui é executada a interação com a porta de série, havendo controle de erros, fluxos de dados, stuffing e destuffing de pacotes.

Estrutura do código

Aqui será apresentada uma descrição da estrutura do código implementado no projeto:

• **E_main.c e R_main.c:** são usados para chamar as funções principais do emissor e do receptor respectivamente;

• **E_Application_layer.c e E_Application_layer.h:** estão incluídos os headers da camada de ligação de dados do Emissor;

• **R_Application_layer.c e R_Application_layer.h:** estão incluídos os headers da camada de ligação de dados do Recetor.

Estes últimos 4 ficheiros têm a finalidade de fazer a ligação entre receptor e emissor. Nessas duas camadas, encontra-se a implementação das funções principais:

• **ficheiro():** No emissor, a função ficheiro() está implementada para saber qual é nome do ficheiro que é dado no argumento passado na shell. No recetor, a função ficheiro() guarda toda a informação dos tramas do novo ficheiro criado;

• **stat():** determina o tamanho, em bytes, do ficheiro que possui um campo **off_t st_size**;

• **call_llopen():** chama a função llopen();

• **call_llclose():** chama a função llclose();

• **emissor() e recetor():** enviam os pacotes de controlo de dados. O recetor usa o llread() para os receber e o emissor usa o llwrite() para os escrever no novo ficheiro;

A função **emissor()** cria o pacote de dados com o campo de controlo de START, que guarda o tamanho do ficheiro e o nome do ficheiro. Para tal, foi criada a função **PacoteControlo()**. Sabendo que cada bloco é um octeto, temos que perceber quantos blocos irão ser ocupados, daí a existência da função **blocos()**.

Para além disso, sabendo que um pacote envia, no máximo 256 bytes em cada trama de informação e que o máximo de bytes de um bloco é 255 bytes, é necessário saber o número de pacotes a enviar, tendo sido criada, para tal, a função **PacoteEnviar()**.

Conhecendo o número de pacotes, podemos fazer um ciclo while que criará um pacote de dados quando houver pacotes a enviar. Assim, implementamos a função **PacoteDados()**. Esta devolverá um pacote de dados com a posição inicial e a final da informação que enviamos. No final, o pacote de controlo END é enviado.

A função **recetor()** segue os mesmos passos do **emissor()**. No entanto, existe uma diferença: no emissor o pacote irá ser criado, no recetor irá existir uma função chamada analisarPacote(), que verifica se o pacote é START,END ou DADOS. Caso seja START, é possível obter o tamanho ou o nome do ficheiro; caso seja END, termina o pacote; caso seja DADOS, se a sequência desses pacotes for correta, toda a informação irá ser guardada no apontador *mensagem, que se encontra dentro de um ciclo while que controla quantos pacotes faltam receber, determinado pela função pacotesReceber().

Os ficheiros **E_link_layer.c**,**R_link_layer.c** e os seus headers contêm a implementação da camada ligação de dados. Nestes ficheiros contém as funções **llopen()**, **llclose(**), **llwrite()** e **llread(**).

A função **llopen()** contém as funções auxiliares **TramaSupervisor()** e **verificarTramaS()**. A **TramaSupervisor()** cria uma trama, que recebe um controlo C como argumento e a função **verificarTramaS()** verifica, não só se a trama de controlo recebida é a trama que se está á espera de obter, como também se o BCC1 está correto. Para ler a trama, é necessário o uso da função **lerTrama()**, que lê qualquer tipo de trama seja ela de

Supervisão ou de Informação. A leitura começa quando a flag F, com valor 0x7E, é lida e termina quando se obtém o mesmo da flag F.

A função **llclose()** é implementada de forma semelhante à da **llopen(**) com ajuda das funções **TramaSupervisor()** e **verificarTramaS()**.

A função llwrite() contém a implementação das funções **Tramal()** e **stuffing(**). A primeira função irá criar uma trama de informação, adicionando os cabeçalhos ao buffer e o pacote de dados ou de controlo que é recebido como argumento na função **llwrite()**, calculando também o BCC1 e o BCC2. Por sua vez, a segunda função será fazer o stuffing.

O stuffing começa a funcionar quando um octeto tem como valor igual à nossa flag F. Esse octeto irá ser substituído por dois octetos com o valor 0x7D e 0x5D, respectivamente. De seguida, são enviados para a porta, esperando por uma trama de controlo para continuar com este processo. Se a trama receber um controlo de RR, então irá retornar o número de bits enviados após o stuffing, senão envia de novo a trama até o número de tentativas for excedido.

A função **llread()** aguarda a leitura da trama. Quando a função **lerTrama()** acaba de ler a trama, irá retornar e verificar se o BCC1 está correto. Se tudo correr bem, será realizado o seu **destuffing()** (processo oposto do **stuffing()**, explicado no parágrafo anterior) e verificará se o BCC2 está correto. Caso tal aconteça, o pacote é guardado num buffer, que é o argumento na função **llread()**, e envia para a trama o controlo RR. Caso contrário, envia o controlo REJ.

Casos de uso principais

A aplicação deste trabalho resulta da transferência do ficheiro e da inserção de dados por parte do utilizador. Este deve inserir a porta série a usar, o nome do ficheiro a enviar e o nome do ficheiro a receber.

A **Transmissão de dados através da porta de série** resulta com a seguinte sequência:

1. Emissor escolhe o ficheiro a enviar ao recetor;

2. Configuração e estabelecimento da ligação entre os dois computadores;

3. Emissor envia os dados para o recetor de trama a trama;

4. Recetor recebe os dados e os guarda num ficheiro com o nome que foi dado no comando da shell;

5. Termina a ligação;

Protocolo de ligação lógica

Função llopen()

A função **llopen()** estabelece a ligação entre o receptor e o emissor. É esperado que o recetor receba uma trama de controlo SET, para enviar o comando UA. Caso isto aconteça,

a ligação foi estabelecida com sucesso. Por parte do emissor, é esperado que a trama de controlo SET seja enviada, aguardando pela resposta do recetor. Se esta resposta não chegar durante 3 segundos, o SET é enviado outra vez, podendo-se repetir até 3 vezes. Se a resposta não chegar mesmo após estas 3 tentativas, o programa termina.

Esta função recebe a porta que está a ser utilizada como argumento e retorna um inteiro positivo se correr tudo bem ou negativo se ocorrer algum erro.

Função llclose()

A função **llclose()** termina a ligação entre o emissor e o recetor. O emissor envia o comando DISC pela porta de série e aguarda pela resposta do recetor DISC, enviando depois a resposta UA. Os alarmes verificam se este procedimento acontece sem problemas com um número máximo de tentativas de conexão e um tempo limite.

Função llread()

A função **llread()** faz a leitura dos pacotes de informação, dados pela a porta de série. Se a mensagem não tiver erros, é guardada e o comando RR é transmitido. Caso contrário, envia o comando REJ, através da porta de série.

Função llwrite()

A função **llwrite()** recebe como argumento um buffer a transmitir e envia a trama ao recetor já com o stuffing concluído, aguardando, depois, por uma resposta. Se não for recebida num intervalo de tempo definido, a trama é enviada outra vez. Caso a resposta seja recebida e seja a trama RR, então a função termina com sucesso. Caso isto não aconteça, a resposta será REJ, o que significa que a mensagem não foi transmitida corretamente e a trama deve ser enviada de novo.

Protocolo de aplicação

A camada de aplicação é responsável por enviar e receber pacotes e ficheiros. Os pacotes podem ser de dados, controlo de inicial ou final. A diferença de pacote de controlo de inicial e de controlo final está no primeiro byte do pacote.

Pacotes de controlo

Os pacotes de controlo marcam o início e o fim do envio ou da leitura de um ficheiro. Para o pacote Controlo ser criado, implementamos a função **PacoteControlo()**, que irá chamar a função **llwirte()**.

Pacotes de dados

Os pacotes de dados tem como objetivo transportar as tramas de informação, ou seja, os fragmentos do ficheiro a transferir. Foi criada a função **PacoteDados()**, que cria os

pacotes de dados da camada de ligação de dados que faz o envio. Esta função invoca a função **llread(),** uma vez que o recetor tem de receber a trama de informação.

Validação

A validação do programa foi feita pelos os seguintes testes:
• Transferência de ficheiros com tamanhos diferentes;
• Interrupção da ligação entre as portas de série;

Eficiência do protocolo de ligação de dados

Tamanho Trama I	C	R	S=R/C
128	28800 bit/s	4703.323 bit/s	0.1633
256		8147.667 bit/s	0.2829
512		9624.142 bit/s	0.3342
128	38400 bit/s	4727.714 bit/s	0.1231
256		1299.347 bit/s	0.0338
512		12229.010	0.3184
128	57600 bit/s	4996.396 bit/s	0.0867
256		8303.780 bit/s	0.1442
512		13537.471 bit/s	0.2350

Tabela 1: Eficiência

Através dos dados obtidos na tabela 1 , é possível afirmar que, quanto maior for a capacidade de ligação, menor é a sua eficiência.

Conclusão

Foi proposto ao grupo a realização de um projeto que consistia essencialmente na transferência de ficheiros entre máquinas através de uma ligação com as portas de séries.

Relativamente ao trabalho, o grupo compreendeu bem todos os pontos abordados no guião estando este dividido em duas grandes camadas, como foi abordado anteriormente na secção da Arquitetura. As camadas são independentes e possuem uma ligação unidirecional, sendo que a de aplicação exerce o controlo perante a de ligação de dados, não se sucedendo o contrário. O cabeçalho dos pacotes a transportar nas tramas de informação não é processado na camada de ligação de dados, mas sim invisível para ela.

Nesta camada, não existe distinção entre pacotes de dados nem de controlo e as numerações dos pacotes são de análise desnecessária.

No que diz respeito à camada da aplicação, esta não conhece as propriedades da camada de ligação de dados porém acede aos serviços por ela fornecidos. As estruturas, o mecanismo de delineação, proteção e retransmissões de tramas são desconhecidos por esta camada, como era abordado no guião do trabalho, assim como a existência de stuffing e destuffing são todos processos tratados ao nível da camada de ligação de dados.

Com este trabalho, foram alcançados os objetivos pretendidos e permitiu uma boa consolidação de conhecimentos, tanto da matéria lecionada nas aulas teóricas, como também na aplicação da mesma nas aulas laboratoriais.

Anexo Código fonte

E_Application_layer.c

```
#include "../include/E_Application_layer.h"
#include "../include/E_link_layer.h"



#define M 0xFF //255 bites

#define BAUDRATE B38400


 struct termios oldtio;
 struct termios newtio;

unsigned char N = 0x00; //numero sequencia do pactote de dados 255

/*Variáveis para os ficheiros*/
unsigned char* nameFile; int sizefile;
int sizeNameFile;
unsigned char* mensage_content;

void ficheiro(char *file){
  sizeNameFile = strlen(file);
  nameFile = (unsigned char*) malloc(sizeNameFile);
  nameFile = (unsigned char*) file;

  struct stat sfile; // struct informaçoes sobre o file

  if (stat(file, &sfile) == -1) { //se informaçao sobre o file nao
passou da erro
```

```
      perror("stat");
      exit(EXIT_FAILURE);
  }

  sizefile = sfile.st_size;
  int tamanho = sizefile;

      mensage_content  = (unsigned char*) malloc(tamanho);

  FILE *file_Emissor;
  file_Emissor = fopen(file, "rb");

  if(file_Emissor==NULL){
      printf("File not found existe\n");
      exit(-1);
  }

  printf("%ld tamanho do ficheiro\n", (long int)sizefile);

  fread(mensage_content, sizefile+1, sizeof(mensage_content),
file_Emissor);

}

void call_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;
  newtio.c_lflag = 0;
  newtio.c_cc[VTIME] = 1;
  newtio.c_cc[VMIN] = 0;

  tcflush(fd, TCIOFLUSH);
  if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
      perror("tcsetattr");
      exit(-1);
  }
  printf("New termios structure set\n");
```

```
  if(llopen(fd)==1) printf("Ligação estabelecidda com sucesso\n");
  else{
      printf("Ligação falhada\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
      }
      close(fd);
      exit(-1);
  }
}

void call_llclose(int fd){
  if(llclose(fd)==1){
   printf("Terminada com sucesso\n");
   sleep(1);
   if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
   }
   close(fd);
  }
  else{
      printf("Terminada sem sucesso\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
      }
      close(fd);
      exit(-1);
  }
}

int blocos(int size, int *resto){ // Quantos blocos faltam a ocupar
maximo de um bloco 255 bites
  long int tamanho = (long int) size;
  int i=1;
  while(tamanho-255 > 0){
      i++;
      tamanho-=255;
  }
  *resto=tamanho;
  return i;
}
```

```c
int V1(unsigned char* pacote, int start, int l1, int rest){ //V1 numero de octetos indicado em L
  int i;
  for(i=start; i<(start+l1-1); i++){
      pacote[i]=M;
  }
  pacote[i++]=(unsigned char)rest;
  return i;
}

unsigned char* PacoteControlo(unsigned char controlo, int *sizePackage){
//guarda o tamanho do ficheiro
  unsigned char* package;
  int i=0, resto;
  int l1 = blocos(sizefile,&resto); //tamanho de octetos

  package = (unsigned char*)malloc(5+l1+sizeNameFile);

  package[i++]=controlo;
  package[i++]=T1;
  package[i++]=(unsigned char)l1;
  i = V1(package,i,l1,resto);
  package[i++]=T2;
  package[i++]=(unsigned char)sizeNameFile;
  memcpy(package+i, nameFile, sizeNameFile);
  i=i+sizeNameFile;

  *sizePackage=i;
  return package;
}

int PacoteEnviar(int *rest){ //numero de pacotes a enviar
  long int  tamanho= (long int) sizefile;
  int i=1;
  while(tamanho-256 > 0){
      i++;
      tamanho-=256;
  }
  *rest = tamanho;
  return i;
}

void emissor(int fd){
  unsigned char* package;
```

```
  int sizePackage;

  package =PacoteControlo(START, &sizePackage);



  int res=llwrite(fd,package,sizePackage);
  if(res>0){ //START
      printf("Trama START enviada com sucesso, com %d bits\n", res);
  }
  else{
      printf("Trama START enviada sem sucesso\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
      }
      close(fd);
      exit(-1);
  }

  int pacoteEnviar, l1;
  pacoteEnviar = PacoteEnviar(&l1);

  int start=0, end=256;
  while(pacoteEnviar>0){
      if(pacoteEnviar==1){
      package= PacoteDados(mensage_content, start, (long int)sizefile,
0x00, (unsigned char)l1, &sizePackage);
      res=llwrite(fd,package,sizePackage);
      if(res>0){
      printf("Trama I %d enviada com sucesso, com %d
bits\n",(int)package[1], res);
      }
      else{
      printf("Trama I recebida sem sucesso\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
            perror("tcsetattr");
            exit(-1);
      }
      close(fd);
      exit(-1);
      }
      }
      else{
```

```
      package= PacoteDados(mensage_content, start, end, 0x01, 0x00,
&sizePackage);
      res=llwrite(fd,package,sizePackage);
      if(res>0){
      printf("Trama %d enviada com sucesso, com %d
bits\n",(int)package[1] ,res);
      }
      else{
      printf("Trama  recebida sem sucesso\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
      }
      close(fd);
      exit(-1);
      }
      start=end;
      end+=256;
      }
      pacoteEnviar--;
  }

  package = PacoteControlo(END, &sizePackage);
  res=llwrite(fd,package,sizePackage);
  if(res>0){ //START
      printf("Trama END enviada com sucesso, com %d bits\n", res);
  }
  else{
      printf("Trama END enviada sem sucesso\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
      }
      close(fd);
      exit(-1);
  }
}

unsigned char* PacoteDados(unsigned char* mensagem, int start,long int
end, unsigned char l2, unsigned char l1, int *sizePacote){
  unsigned char* pacote;
  int k = (256 * (int)l2) + (int)l1;
  pacote = (unsigned char*) malloc (4+k);
```

```
  int i =0;
  pacote[i++]=DADOS;
  pacote[i++]=N;
  pacote[i++]=l2;
  pacote[i++]=l1;

  for(int j=start; j<end; j++){
      pacote[i++]=mensagem[j];
  }

  *sizePacote=i;

  N+=0x01;
  return pacote;
}
```

E_link_layer.c

```
#include "../include/E_link_layer.h"


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

#define FALSE 0
#define TRUE 1

/*flags para o alarme*/
int flag_alarme=0, conta_alarme=0;

/*flags para controlar o que é lido*/
volatile int STOP=FALSE;
int res;
unsigned char buf[256];

/*flags para controlar os estados em tramas supervisão*/
int stateInicio, stateFim, stateMedio;
```

```c
/*bits de controlo em envio*/
int Ns=0;

void alarme(){
  printf("alarme #%d\n", conta_alarme+1);
  flag_alarme = 1;
  conta_alarme++;
}

int llopen(int fd){
  (void) signal(SIGALRM, alarme);


      unsigned char set[5];
      TramaSupervisor(set, SET);

      conta_alarme=0;

      while (conta_alarme<3) {
      fflush(stdout);
      res=write(fd,set,5);

      flag_alarme=0;
      alarm(3);

      if(verificarTramaS(fd,UA)==1) return 1;
      }

      return -1;
}

void TramaSupervisor(unsigned char* trama, unsigned char comando){
  trama[0]=F;
  trama[1]=A;
  trama[2]=comando;
  trama[3]=(A^comando);
  trama[4]=F;
}

unsigned char* lerTrama(int fd){
  unsigned char* rec = (unsigned char*)malloc(5);
  int i=0;
  STOP=FALSE;
  stateInicio=1, stateFim=0, stateMedio=0;
```

```
  while (STOP==FALSE) {
      res=read(fd,buf,1); //leitura byte a byte
      if(res>0){
      if(buf[0]==F && stateInicio){ //inicio da leitura
      rec[i]=buf[0];
      stateInicio=0;
      stateMedio=1;
      i++;
      }
      else if(buf[0]==F && stateMedio){ //fim da leitura
      rec[i]=buf[0];
      stateMedio=0;
      stateFim=1;
      i++;
      }
      else if(stateMedio){ //estado no meio
      rec[i]=buf[0];
      i++;
      }
      }
      else if(stateFim || flag_alarme){
      STOP=TRUE;
      }
  }
  fflush(stdout); //buffer fica limpo
  return rec;
}

int verificarTramaS(int fd, unsigned char comando){ //verifica o seu
bbc1 e se a trama esta correta
  unsigned char* rec=lerTrama(fd);
  alarm(0);
  if(rec[3]==(rec[1]^rec[2]) && rec[2]==comando) return 1;
  else return -1;
}

int llclose(int fd){
  unsigned char disc[5];
  TramaSupervisor(disc, DISC);
  unsigned char ua[5];
  TramaSupervisor(ua,UA);

  conta_alarme=0;

  while (conta_alarme<3) {
      flag_alarme=0;
```

```
      alarm(3);
      fflush(stdout);
      res=write(fd,disc,5);


      if(verificarTramaS(fd,DISC)==1) {
      fflush(stdout);
      write(fd,ua,5);
      return 1;
      }
  }

  return -1;
}

unsigned char BCC2(unsigned char* package, int sizePackage){ //bbc2
campo de proteçao de dados
  unsigned char r = package[0];
  for(int i=1; i<sizePackage; i++){
      r=r^package[i];
  }
  return r;
}

unsigned char* TramaI(unsigned char* package, int sizeP, int *sizeI){
  unsigned char* trama;
  int i=0;
  trama=(unsigned char*)malloc(sizeP+6);
  trama[i++]=F;
  trama[i++]=A;

  switch (Ns) {
      case 0:
      trama[i++]=C0;
      break;
      case 1:
      trama[i++]=C1; //64
      break;
      default:
      printf("Erro no bit\n");
      exit(-1);
      break;
  }

  trama[i++] = (A^trama[2]);
  memcpy(trama+i, package, sizeP);
```

```
  i = i+sizeP;
  trama[i++] = BCC2(package, sizeP);
  trama[i++] = F;

  *sizeI = i;

  return trama;
}

int llwrite(int fd, unsigned char* package, int sizePackage){
  if(conta_alarme==3) return -1;
  int length;
  unsigned char* trama = TramaI(package, sizePackage, &length);

  conta_alarme=0;
  int reject=0,sizeTramaInformacao;

  unsigned char* tramaI = stuffing(trama, length, &sizeTramaInformacao);

  while(conta_alarme<3 || reject){
      //printf("%d conta_alarme || %d rejeitar\n", conta_alarme,
rejeitar);
      if(conta_alarme==3) return -1;
      fflush(stdout);
      int res1=write(fd,tramaI,sizeTramaInformacao);
      //printf("%d envidos\n",res1);
      flag_alarme=0;
      alarm(3);

      switch (Ns) {
      case 0:
      if(verificarTramaS(fd,RR1)==1){
            Ns=1;
            return res1;
      }
      else if(verificarTramaS(fd,REJ0)==1){
            printf("REJ0 recebido\n");
            reject=1;
      }
      break;
      case 1:
      if(verificarTramaS(fd,RR0)==1){
            Ns=0;
            return res1;
      }
      else if(verificarTramaS(fd,REJ1)==1){
```

```
            printf("REJ1 recebido\n");
            reject=1;
      }
      break;
      default:
      printf("Erro\n");
      exit(-1);
      break;
      }
  }
  return -1;
}

unsigned char* stuffing(unsigned char* trama, int length, int
*sizeTramaI){
  unsigned char* tramaStuff;
  tramaStuff = (unsigned char*) malloc(2*(length+1)+5);
  int i=0;

  tramaStuff[i++]=F;
  tramaStuff[i++]=A;

  switch (Ns) {
      case 0:
      tramaStuff[i++]=C0;
      break;
      case 1:
      tramaStuff[i++]=C1;
      break;
      default:
      printf("Erro no bit\n");
      exit(-1);
      break;
  }
  tramaStuff[i++] = (A^tramaStuff[2]);

  for(int j=0; j<length; j++){
      if(trama[j]==SETEE){
      tramaStuff[i++]=SETED;
      tramaStuff[i++]=CINCOE;
      }
      else if(trama[j]==SETED){
      tramaStuff[i++]=SETED;
      tramaStuff[i++]=CINCOD;
      }
      else{
```

```
      tramaStuff[i++]=trama[j];
      }
  }

  tramaStuff[i++] = F;
  *sizeTramaI = i;
  return tramaStuff;
}
```

E_main.c

```
#include "../include/E_Application_layer.h"



int fd;

int main(int argc, char *argv[]) {

  if ( (argc < 2) ||
           ((strcmp("/dev/ttyS10", argv[1])!=0) &&
           (strcmp("/dev/ttyS11", argv[1])!=0) )) {
      printf("Usage:SerialPort,ex:/dev/ttyS10\n");
      exit(-1);
  }

   /* if ( (argc < 2) ||
      ((strcmp("/dev/ttyS4", argv[1])!=0) &&
           (strcmp("/dev/ttyS0", argv[1])!=0) )) {
      printf("Usage:SerialPort,ex:/dev/ttyS4\n");
      exit(-1);
      }*/
  else if (argc < 3){
      printf("Numero de argumentos errado\n");
      exit(-1);
  }

  fd = open(argv[1], O_RDWR | O_NOCTTY );
  if (fd <0) {perror(argv[1]); exit(-1); }

  call_llopen(fd);
```

```
  ficheiro(argv[2]);

  emissor(fd);

  call_llclose(fd);

  return 0;
}
```

R_Application_layer.c

```
#include "../include/R_Application_layer.h"
#include "../include/R_link_layer.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 <unistd.h>

#define BAUDRATE B38400
#define _POSIX_SOURCE 1
#define FALSE 0
#define TRUE 1

#define MAXSIZE 1024

/*Mensagem recebida*/
unsigned char* mensage;
int bytes=0; //bytes do ficheiro
int packagesReceive=0;//pacotes que receberam

/*Informação recebida do ficheiro*/
char *nameFile;
long int sizeFile;

int nFlag=0;
```

```
struct termios oldtio;
struct termios newtio;

void call_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;
  newtio.c_lflag = 0;
  newtio.c_cc[VTIME] = 1;
  newtio.c_cc[VMIN] = 0;

  tcflush(fd, TCIOFLUSH);
  if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {
      perror("tcsetattr");
      exit(-1);
  }
  printf("New termios structure set\n");

  if(llopen(fd)==1) printf("Ligação estabelecidda com sucesso\n");
  else{
      printf("Ligação falhada\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
      }
      close(fd);
      exit(-1);
  }
}

void call_llclose(int fd){
  if(llclose(fd)==1) {
      printf("Terminada com sucesso\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
```

```
      }
      close(fd);
  }
  else{
      printf("Terminada sem sucesso\n");
      sleep(1);
      if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
      perror("tcsetattr");
      exit(-1);
      }
      close(fd);
      exit(-1);
  }
}

void pacotesReceber(){ // numero de pacotes a receber
  long int tmp = (long int) sizeFile;
  int r=1;
  while(tmp-256 > 0){
      r++;
      tmp-=256;
  }
  packagesReceive=r;
  return;
}

void recetor(int fd,char *file){
  unsigned char* pacote = (unsigned char*) malloc (MAXSIZE);
  int sizePacote = llread(fd, pacote);
  /*Receção da trama Start*/
      TipoPacote(pacote, sizePacote);

  bytes=0;
  pacotesReceber();

  printf("%d pacotes a receber\n", packagesReceive);
  while(packagesReceive>0){
      //printf("%d pacotes em falta\n",packagesReceive);
      free(pacote);
      pacote = (unsigned char*) malloc (MAXSIZE);
      sizePacote = llread(fd, pacote);
      //printf("%d -> sizePacote\n", sizePacote);
      if (sizePacote>0){
      if (TipoPacote(pacote, sizePacote)) packagesReceive--;
      }
      else{
```

```
      //printf("Entrei aqui no 0\n");
      continue;
      }
  }
  printf("%d bytes do ficheiro\n", bytes);
  ficheiro(file);

  free(pacote);
  pacote = (unsigned char*) malloc (MAXSIZE);
  sizePacote = llread(fd, pacote);
  /*Receção trama END*/
  TipoPacote(pacote, sizePacote);
}

int TipoPacote(unsigned char* pacote, int sizePacote){ //verifica se é
Start END DADOS
  unsigned char c = pacote[0];
  switch (c) {
      case START:
      pacoteStart(pacote, sizePacote);
      printf("Trama START bem recebida\n");
      return 1;
      break;
      case END:
      printf("Trama END bem recebida\n");
      return 1;
      break;
      case DADOS:
      if(nFlag==(int)pacote[1]){
      nFlag++;
      if(nFlag==256){nFlag=0;}
      pacoteDados(pacote,sizePacote);
      printf("Trama I bem recebida com %d bytes, pacote %d\n",
sizePacote, (int)pacote[1]);
      return 1;}
      break;
  }
      return 0;
}

void pacoteStart(unsigned char* pacote, int sizePacote){ // se o pacote
tiver o comando Start
  int tamanho = 0x00;
  int index=0;
  int packages;
  if(pacote[1]==T1){ //Tamaho do ficheiro
```

```
      packages = (int)pacote[2];
      for(int i=3; i<3+packages; i++){
      tamanho+=pacote[i];
      }
      sizeFile = (long int) tamanho;
      index=3+packages;
  }

  int lengthName, j=0 ;
  if(pacote[index++]==T2){//Nome do ficheiro;
      lengthName = (int)pacote[index++];
      nameFile= (char*)malloc(lengthName);
      for(int i=index; i<index+lengthName; i++){
      nameFile[j++]=(char)pacote[i];
      }
  }

  printf("tamanho ficheiro: %ld\n", sizeFile);
  printf("nome do ficheiro: %s\n", nameFile);

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

}

void pacoteDados(unsigned char* pacote, int sizePacote){
  int end = (256 * (int)pacote[2]) + (int)pacote[3];
  int j=4;

  for(int i=bytes; i<bytes+end; i++){
      mensage[i]=pacote[j++];
  }
  bytes = bytes + end;
  return;
}

void ficheiro(char *file){
  FILE *fp;
  fp=fopen(file,"wb");

  if(fp==NULL){
      printf("Erro na criação de ficheiro\n");
      exit(-1);
  }
  fwrite(mensage, bytes, sizeof(mensage),fp);

}
```

R_link_layer.c

```c
#include "../include/R_link_layer.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 <unistd.h>


#define FALSE 0
#define TRUE 1

#define MAXSIZE 1024

/*flags para controlar o que é lido*/
volatile int STOP=FALSE;
int res;
unsigned char buf[256];

/*flags para controlar os estados em tramas supervisão*/
int stateInicio, stateFim, stateMedio;

/*bits de controlo em envio*/
int Nr=0;

/*tamanho da trama lida*/
int size;

int llopen(int fd){

  unsigned char ua[5];
  TramaSupervisor(ua,UA);

  if(verificarTramaS(fd,SET)==1){
      fflush(stdout);
      res=write(fd,ua,5);
      return 1;
```

```
  }
  return -1;
}

int verificarTramaS(int fd, unsigned char comando){ //verifica se a
trama de controlo recebida esra certa
  unsigned char *rec;
  rec = lerTrama(fd);

  if(rec[3]==(rec[1]^rec[2]) && rec[2]==comando) return 1;
  else return -1;
}

void TramaSupervisor(unsigned char* trama, unsigned char comando){ //
cria um trama supervisor
  trama[0]=FLAG_RCV;
  trama[1]=A_RCV;
  trama[2]=comando;
  trama[3]=(A_RCV^comando);
  trama[4]=FLAG_RCV;
}

int llclose(int fd){
  unsigned char disc[5];
  TramaSupervisor(disc,DISC);

  if(verificarTramaS(fd,DISC)==1){
      fflush(stdout);
      res=write(fd,disc,5);
  }

  if(verificarTramaS(fd,UA)==1){
      return 1;
  }

  return -1;
}

unsigned char* lerTrama(int fd){
  unsigned char *rec;
  rec=(unsigned char*)malloc(MAXSIZE);
  int i=0;
  STOP=FALSE;
  stateInicio=1, stateFim=0, stateMedio=0;

  while (STOP==FALSE) {
```

```
      res=read(fd,buf,1);
      if(res>0){
      if(buf[0]==FLAG_RCV && stateInicio){ //inicio da leitura
      rec[i]=buf[0];
      stateInicio=0;
      stateMedio=1;
      i++;
      }
      else if(buf[0]==FLAG_RCV && stateMedio){ //fim da leitura
      rec[i]=buf[0];
      stateMedio=0;
      stateFim=1;
      i++;
      }
      else if(stateMedio){ //estado no meio
      rec[i]=buf[0];
      i++;
      }
      }
      else if(stateFim){
      STOP=TRUE;
      }
  }
  //printf("%d recebidos\n", i);
  size=i;
  fflush(stdout);
  return rec;
}


int llread(int fd, unsigned char* buffer){
  unsigned char* tmp;
  unsigned char c;
  tmp = lerTrama(fd);
  c=tmp[2];

  int sizeTramaI, sizePacote;
  unsigned char* tmp1;
  unsigned char* pacote;
  unsigned char bcc;
  if(tmp[3]==(tmp[1]^tmp[2])) {
      tmp1 = destuffing(tmp, &sizeTramaI);
      bcc = tmp1[sizeTramaI-2];
      pacote=Headers(tmp1,sizeTramaI, &sizePacote);
  }
```

```
unsigned char* controlo;

if(BCC2(pacote, sizePacote, bcc)){
    /*Mandar a mensagem para buffer*/
    memcpy(buffer,pacote, sizePacote);
    switch (Nr) {
    case 0:
    if(c==C0){
          Nr=1;
          controlo = (unsigned char*) malloc(5);
    TramaSupervisor(controlo, RR1);
          fflush(stdout);
          write(fd, controlo, 5);
    }
    else if(c==C1){
          controlo = (unsigned char*) malloc(5);
    TramaSupervisor(controlo, RR0);
          fflush(stdout);
          write(fd, controlo, 5);
    }
    break;
    case 1:
    if(c==C1){
          Nr=0;
          controlo = (unsigned char*) malloc(5);
          TramaSupervisor(controlo, RR0);
          fflush(stdout);
          write(fd, controlo, 5);
    }
    else if(c==C0){
          controlo = (unsigned char*) malloc(5);
          TramaSupervisor(controlo, RR1);
          fflush(stdout);
          write(fd, controlo, 5);
    }
    break;
    }
}
else{
    switch (Nr) {
    case 0:
    controlo = (unsigned char*) malloc(5);
    TramaSupervisor(controlo, REJ0);
    fflush(stdout);
    write(fd, controlo, 5);
    //printf("Mandado REJ0\n");
```

```
      return 0;
      break;
      case 1:
      controlo = (unsigned char*) malloc(5);
      TramaSupervisor(controlo, REJ1);
      fflush(stdout);
      write(fd, controlo, 5);
      //printf("Mandado REJ1\n");
      return 0;
      break;
      }
  }

  return sizePacote;
}

unsigned char* Headers(unsigned char* tramaI, int sizeTramaI, int
*sizePacote){ //colocar o pacote so com informaçao util
  unsigned char* pacote;
  pacote = (unsigned char*) malloc (sizeTramaI-6);

  int j=0;
  for(int i=4; i<sizeTramaI-2; i++){
      pacote[j++]=tramaI[i];
  }

  *sizePacote = j;
  return pacote;
}

int BCC2(unsigned char* pacote, int sizePacote, unsigned char bcc){
//campo de proteçao de dados
  /*cacular BCC2*/
  unsigned char bcc2 = pacote[0];
  for(int i=1; i<sizePacote; i++)
      bcc2 = bcc2 ^ pacote[i];

  return (bcc2==bcc);
}

unsigned char* destuffing(unsigned char* tramaI, int *sizeTrama){
  unsigned char* r;
  r=(unsigned char*)malloc(MAXSIZE);
  /*i=4 porque as tramas de informação vieram com headers
  acaba em size-1 por causa da flag final*/
  int i=4, j=0;
```

```
  while (i<size-1) {
      if(tramaI[i]==SETED && tramaI[i+1]==CINCOE){
      r[j]=SETEE;
      i+=2;
      j++;
      }
      else if(tramaI[i]==SETED && tramaI[i+1]==CINCOD){
      r[j]=SETED;
      i+=2;
      j++;
      }
      else{
      r[j]=tramaI[i];
      j++;
      i++;
      }
  }
  *sizeTrama = j;
  return r;
}
```

R_main.c

```
#include "../include/R_Application_layer.h"



int fd;

int main(int argc, char *argv[]) {
  if ( (argc < 2) ||
            ((strcmp("/dev/ttyS10", argv[1])!=0) &&
            (strcmp("/dev/ttyS11", argv[1])!=0) )) {
      printf("Usage: SerialPort,ex:/dev/ttyS11\n");
      exit(-1);
      }
      /*if ( (argc < 2) ||
      ((strcmp("/dev/ttyS4", argv[1])!=0) &&
            (strcmp("/dev/ttyS0", argv[1])!=0) )) {
      printf("Usage:SerialPort,ex:/dev/ttyS0\n");
      exit(-1);
      }*/
```

```
  fd = open(argv[1], O_RDWR | O_NOCTTY );
  if (fd <0) {perror(argv[1]); exit(-1); }

  call_llopen(fd);
  recetor(fd,argv[2]);
  call ll close(fd);

  return 0;
}
```