Protocolo de Ligação de Dados

Relatório

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Redes de Computadores

Jorge Filipe Monteiro Lima - 201000649
Nuno Filipe Dinis Cruz - 201004232
Vasco Fernandes Gonçalves- 201006652

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

11 de Novembro de 2013

1 Introdução

Neste relatório iremos descrever o nosso primeiro trabalho laboratorial, que teve como objectivos implementar um protocolo de ligação de dados, ou seja, um serviço de comunicação de dados fiável entre dois sistemas ligados por canal de transmissão, neste caso, um cabo de série. Usando Linux como ambiente de desenvolvimento, a linguagem usada foi o C.

Foi também necessário testar o protocolo com uma aplicação de transferência de ficheiros.

2 Arquitectura

O nosso trabalho está dividido em 3 blocos funcionais, de acordo com o nome dos ficheiros .c:

- Write - bloco com as funções necessárias para o *transmitter*, nomeadamente o envio de pacotes de controlo e de dados;
- Read - bloco com as funções necessárias para o *receiver*, nomeadamente a recepção de pacotes de controlo e de dados;
- Protocol - bloco com funções de envio/recepção de tramas de informação e de supervisão.

Os primeiros dois tratam da parte da aplicação, enquanto que o último trata da camada de mais baixo nível, o protocolo.

3 Estrutura do código

O código está dividido em 3 ficheiros, como descrito na secção anterior. As principais funções de cada um são as seguintes:

Read.h

```
//Maquina de estados que trata do parsing dos pacotes
int packetStateMachine(char *buffer);

//Chama a funcao do protocolo que trata do establecimento ou
    terminacao da conexao por parte do receiver, respectivamente
int llopen(char *porta);
int llclose(int fd);

//Chama a funcao do protocolo que efectua a leitura de dados
int llread(int fd, char *buffer);
```

Write.h

```
//Chama a funcao do protocolo que trata do establecimento/
    terminacao da conexao por parte do transmitter,
    respectivamente
int llopen(char *porta);
int llclose(int fd);

//Trata do envio de pacotes de dados
void sendFile(int fd, unsigned int fileSize, char *fileName,
    char *fileContent);

//Abre ficheiro para leitura em binario, devolvendo o tamanho em
     bytes do ficheiro
int openFile(char *fileName, char **fileContent)
```

Protocol.h

```
//Trata do byte stuffing, mexe na variavel global writeBuf[]
int stuffBytes(unsigned int numChars);

//Constroi e envia a trama de controlo com 'c' como campo de
    controlo
void sendControlFrame(int fd, int c);

//Trata de iniciar a conexao entre o transmitter(user=0) e o
    receiver(user=1)
int setConnection(char * porta, int user);

//Chama o readFrame, para leitura de uma trama, e interpreta os
    resultados
int readData(int fd, char * buffer);

//Faz o parsing da trama recebida
int readFrame(int fd, char * buffer, int *res);

//Trata de iniciar a conexao
int disconnect(int fd, int user);

//Funcao auxiliar que constroi e envia um trama de dados com um
    pacote 'packet', chama a funcao sendFrame()
int sendData(int fd, char *packet, int packetChars);

//(Re-)envia a informacao existente em writeBuf ate 3 vezes, e
    chamada na funcao alarm()
void sendFrame();
```

4 Casos de uso principais

Sequências de chamada de funções:
Write(*transmitter*):

- openFile(fileName, & fileContent);
- int fd = llopen(argv[1]);
  - setConnection(porta, 0);
- sendFile(fd, fileSize, fileName, fileContent);
  - sendControlPacket(fd, 1, fileSize, fileName);
    - llwrite(fd, controlPacket, i);
  - sendDataPacket(fd, chunkBuf, CHUNK_SIZE, n); //chamado varias vezes
    - llwrite(fd, packet, dataChars+4); //por causa do cabeçalho
  - sendControlPacket(fd, 2, fileSize, fileName);
    - llwrite(fd, controlPacket, i);
- llclose(fd);
  - disconnect(fd, 0);

Read(*receiver*):

- int fd = llopen(argv[1]);
    - setConnection(porta, 1);
- int res = llread(fd, chunkBuf);
    - readData(fd, buffer);
        * sendControlFrame(...);
- if(res == 0) { if(llclose(fd) <0) return -1; break; }
    - disconnect(fd, 0);

  else if(res != -1) packetStateMachine(chunkBuf);

5 Protocolo de ligação lógica

O protocolo de ligação lógica tem os seguintes aspectos funcionais:

- Envio de tramas de supervisão
- Envio de tramas com *timeout*
- Estabelecimento de ligação
- Construção e envio de tramas de informação
- *Byte stuffing*
- Leitura de tramas
- Extração de pacotes de dados de tramas de informação
- Término de ligação

5.1 Envio de tramas de supervisão

As tramas de supervisão são enviadas com a seguinte função:

```
void sendControlFrame(int fd, int c)
{
    writeBuf[0]=FLAG;
    writeBuf[1]=A;
    writeBuf[2]=c;
    writeBuf[3]=writeBuf[1]^writeBuf[2];
    writeBuf[4]=FLAG;
    writeBufLen=5;
    printf("SENDING COMMAND 0x%x\n", c);
    write(fd, writeBuf, writeBufLen);
}
```

Esta função constrói uma trama de supervisão, sendo que o byte C é passado como parâmetro.

5.2 Envio de tramas com *timeout*

Para enviar tramas com protecção de *timeout* criamos a seguinte função:

```
void sendFrame()
{
    if(conta>0) //incrementa numero de timeouts
        numTimeouts++;
        if(conta++ <= MAX_RETRIES)
        {
            write(globalFD, writeBuf, writeBufLen);
            printf("\tSENDING PACKET\n");
            alarm(3);
        }
        else
        {
            printf("CONNECTION LOST... GIVING UP\n");
            exit(1);
        }
}
```

Esta função é registada no programa como *handler* do *SIGALRM* e de cada vez que é chamada, no máximo o número de vezes definido em *MAX_RETRIES*, é chamada novamente. Quando houver a leitura de uma trama válida, os alarmes pendentes são cancelados.

5.3 Estabelecimento de ligação

Para estabelecer a ligação, a seguinte função é chamada:

```
int setConnection(char * porta, int user)
```

O parâmetro user toma os valores de 0 para o emissor e 1 para o receptor. Esta função configura a porta série e, do lado do emissor, envia a trama de supervisão **SET** e aguarda pela recepção da trama **UA**. Do lado do receptor aguarda a recepção da trama **SET**, envia a trama **UA** e de seguida os dois computadores estão prontos para a transmissão de dados.

5.4 Construção e envio de tramas de informação

Criamos a seguinte função (chamada pelo emissor, na camada da aplicação) para a construção de uma trama de informação:

```
int sendData(int fd, char *packet, int packetChars)
```

O conteúdo do pacote e o seu tamanho são passados da aplicação pelos parâmetros **packet* e *packetChars*. De seguida, é construído o cabeçalho da trama e os dados são inseridos:

```
for (i = 4; i<packetChars + 4; ++i)
{
    writeBuf[i] = packet[i - 4];
    //printf("writeBuf[%i] -> %x\n", i, writeBuf[i]);
    bcc2 ^= writeBuf[i];
}
```

O bcc2 é assim calculado, e os bytes são *stuffed*:

```
int packetCharsAfterStuffing = stuffBytes(packetChars+1);
```

De seguida, a trama é enviada e aguarda-se a resposta (fazendo novas tentativas enquanto não houver uma resposta). Se a resposta for afirmativa (**RR**), a função alterna o C (entre 0 e 2) e retorna, em caso contrário (**RR**) a função volta a ser chamada para efectuar o reenvio.

5.5 *Byte stuffing*

O tratamento dos dados a enviar (*stuffing* é feito pela seguinte função:

```
int stuffBytes(unsigned int numChars)
```

Esta função percorre todos os caracteres dos dados a enviar e, caso encontre os *bytes* **0x7e** ou **0x7d**, passa todos os bytes seguintes uma posição para a direita e insere os caracteres de escape apropriados (**0x5e** ou **0x5d**, respectivamente):

```
memmove(writeBuf+i+1,writeBuf+i,numChars-(i-4));
writeBuf[i]=0x7d;
writeBuf[++i]=<0x5e ou 0x5d>;
```

5.6 Leitura de tramas

Utilizamos uma única função para ler tramas de informação e supervisão:

```
int readFrame(int fd, char * buffer, int *res)
```

Esta função, que representa uma máquina de estados, lê, caracter a caracter, uma trama da porta série e retorna o byte C lido. Sendo uma máquina de estados, esta função só retorna quando tiver sido lida uma trama bem formada, na sua totalidade.

Se se tratar de uma trama de informação, a função escreve os bytes da secção de dados da trama para o array de caracteres *buffer* (passado como parâmetro) e no final escreve também o número de caracteres lidos para o apontador de inteiro *res*. É também alternado o C, entre 0 e 2.

5.7 Extração de pacotes de dados de tramas de informação

De seguida, temos a função que é chamada pelo leitor, na camada de aplicação, para receber pacotes:

```
int readData(int fd, char * buffer)
```

Esta função chama a função anteriormente descrita, faz o tratamento correcto das repostas a dar (conforme a validação do BCC2) e escreve em **buffer* o pacote lido.

É ainda nesta função que recebemos a trama **DISC**, vinda do emissor, e passamos essa informação ao receptor.

5.8 Término de ligação

Por fim, temos a função que termina a ligação:

```
int disconnect(int fd, int user)
```

De acordo com o parâmetro user (0 para emissor, 1 para receptor), envia as tramas **DISC** e **UA** apropriadas e fecha a ligação à porta série.

É também nesta função que mostramos ao utilizador as estatísticas da transmissão.

6 Protocolo de aplicação

O protocolo de aplicação tem os seguintes aspectos funcionais:

- Abertura e fecho de ligação
- Emissor - processamento do ficheiro a enviar
- Emissor - envio de pacotes
- Receptor - recepção de pacotes

6.1 Abertura e fecho de ligação

A aplicação, quando é aberta, recebe por argumento a porta série a utilizar. Com esta informação, é chamada a função:

```
int llopen(char *porta)
```

que simplesmente chama a função **setConnection** do protocolo de ligação lógica. Por sua vez, a função **llclose** chama a função **disconnect**.

6.2 Emissor - processamento do ficheiro a enviar

O emissor recebe por argumento o nome do ficheiro a enviar. Com este nome, chama a seguinte função:

```
int openFile(char *fileName, char **fileContent)
```

que guarda em ***fileContent* o conteúdo do ficheiro e retorna o seu tamanho. De seguida, a função

```
void sendFile(int fd, unsigned int fileSize, char *fileName,
    char *fileContent)
```

divide o ficheiro em *chunks* e envia-os.

6.3 Emissor - envio de pacotes

A função que trata de construir e enviar os pacotes de controlo é a seguinte:

```
void sendControlPacket(int fd, int packetC, unsigned int
    fileSize, char *fileName)
```

Esta função recebe o tipo de pacote a enviar (*Start* ou *End*), constrói-o e envia-o. Quanto aos pacotes de dados, a função é esta:

```
void sendDataPacket(int fd, char *packet, int dataChars,
    unsigned char n)
```

Recebe o *chunk* a enviar no parâmetro **packet*, o seu tamanho e o N (número de referência do pacote), envia o pacote.

6.4 Receptor - recepção de pacotes

O receptor recebe pacotes até o protocolo lhe transmitir um pedido de *disconnect*. No início, cria um ficheiro temporário para onde vai guardando os *chunks* e no final renomeia esse ficheiro para o nome recebido nos pacotes de controlo.

A função que lida com os pacotes é esta:

```
int packetStateMachine(char *buffer)
```

De acordo com o tipo de pacote recebido, armazena as informações sobre o ficheiro em variáveis globais (pacotes de controlo) ou processa os dados recebidos (pacotes de dados).

Os pacotes de dados são verificados, através do **N** (número de referência) e, se não se tratar de um pacote duplicado, armazena o *chunk* para o ficheiro.

7 Validação

O protocolo faz verificações a dois níveis: a nível das tramas com o **BCC1**, que é um xor dos campos do cabeçalho e o **C**, que alterna entre 0 e 2. Estes valores são passados no cabeçalho das tramas.
Verifica ainda, a nível dos pacotes, o **N**, um número de referência que assume valores entre 0 e 255.

Todos os valores estão sincronizados entre emissor e receptor, de forma a poderem ser verificados.

8 Elementos de valorização

Os elementos de valorização implementados foram os seguintes:

Implementação de REJ

As tramas rejeitadas implicam são reenviadas, sem ser necessário esperar pelo próximo timeout:

Do lado do receptor:

```
else if(readC == -1) //invalid bcc2
{
    printf("\nINVALID DATA, SENDING REJECT\n");
    numRejects++;
    sendControlFrame(fd, C_REJ ^ (c<<4)); //passa os 4 bits
       do c para a esquerda
    return -1;
}
```

Do lado do emissor:

```
else //REJECT OR INVALID RESPONSE
{
    numRejects++;
    printf("\tREJECT! RESENDING!!!\n");
    sendData(fd, packet, packetChars); //reenvia
}
```

Verificação da integridade dos dados pela Aplicação

Tamanho do ficheiro recebido

É validado no leitor através dos pacotes de controlo:

```
if(fileSize != ftell(pFile))
    printf("\nERROR TRANSFERING: WRONG FILE SIZE\n");
```

Pacotes de dados duplicados

Validação feita na leitura de pacotes, do lado do leitor, com o número de referência **N**:

```
if(buffer[1]==n) //se nao for um pacote duplicado, grava
   para ficheiro
```

Registo de ocorrências

Ocorrências são gravadas em variáveis globais e escritas ao sair do programa:

```
//IMPRIME ESTATISTICAS
printf("Numero de tramas I: %i\n", numTramas);
printf("Numero de timeouts: %i\n", numTimeouts);
printf("Numero de rejects I: %i\n", numRejects);
```

9 Conclusões

No final deste trabalho concluímos que a definição *à priori* de todas as especificações para o mesmo é extremamente importante e merecedora do tempo nela investido. Só assim podemos evitar constantes obstáculos inerentes da incerteza de como proceder em determinadas situações críticas.

Assim, ficou reforçada a ideia de que para ser engenheiro não basta saber programar, mas também saber planear com exactidão os passos a percorrer no desenvolvimento de qualquer sistema ou aplicação.

Quanto ao tópico do trabalho, pareceu-nos uma boa maneira de abordar, de forma introdutória, a temática das redes de computadores, os problemas mais comuns que podem surgir e como os solucionar.

ANEXO - CÓDIGO FONTE

```c
#pragma once

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

#define BAUDRATE B9600
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FLAG 0x7E
#define A 0x03
#define READER_A 0x01
#define C_SET 0x03
#define C_UA 0x07
#define C_RR 0x05
#define C_REJ 0x01
#define C_DISC 0x0B
#define FALSE 0
#define TRUE 1

#define MAX_RETRIES 3 /* Numero de envios adicionais a efectuar
    em caso de nao haver resposta */
#define CHUNK_SIZE 100

int       conta = 0, //contador para o alarm
          c=0, // campo de controlo a alternar entre 0 e 2
          globalFD, // para passar para signal handler
          writeBufLen,
          numTramas=0,
          numTimeouts=0,
          numRejects=0;

struct termios oldtio; //para restaurar definicoes da porta
    serie no final

char writeBuf[CHUNK_SIZE*2+10], readBuf[1]; //pior caso possivel
    , vao ser enviados o dobro dos caracteres apos stuffing


int stuffBytes(unsigned int numChars);
void sendFrame();
void sendControlFrame(int fd, int c);
int setConnection(char * porta, int user);
int readData(int fd, char * buffer);
int readFrame(int fd, char * buffer, int *res);
int disconnect(int fd, int user);
int sendData(int fd, char *packet, int packetChars);
```

```
#include "protocol.h"

int stuffBytes(unsigned int numChars)
{
  int i=4; //avanca os quatro primeiros (cabecalho da trama)
  for(;i<numChars+4;i++)
  {
    if(writeBuf[i]==0x7e)
    {
      memmove(writeBuf+i+1,writeBuf+i,numChars-(i-4)); //avanca
          todos os caracteres uma posicao
      writeBuf[i]=0x7d;
      writeBuf[++i]=0x5e;
      numChars++; //ha mais um caracter a transmitir
    }
    else if(writeBuf[i]==0x7d)
    {
      memmove(writeBuf+i+1,writeBuf+i,numChars-(i-4)); //avanca
          todos os caracteres uma posicao
      writeBuf[i]=0x7d;
      writeBuf[++i]=0x5d;
      numChars++; //ha mais um caracter a transmitir
    }
  }
  return numChars; //retorna novo numero de caracteres a
      transmitir
}

/* (Re-)envia a informacao existente em writeBuf ate 3 vezes
   e chamada na funcao alarm() */
void sendFrame()
{
  if(conta>0) //incrementa numero de timeouts
    numTimeouts++;
  if(conta++ <= MAX_RETRIES)
  {
    write(globalFD, writeBuf, writeBufLen);
    printf("\tSENDING PACKET\n");
    alarm(3);
  }
  else
  {
    printf("CONNECTION LOST... GIVING UP\n");
    exit(1);
  }
}

//Constroi e envia a trama de controlo com 'c' como campo de
    controlo
void sendControlFrame(int fd, int c)
{
  writeBuf[0]=FLAG;
    writeBuf[1]=A;
    writeBuf[2]=c;
    writeBuf[3]=writeBuf[1]^writeBuf[2];
    writeBuf[4]=FLAG;
  writeBufLen=5;
```

```
  printf("SENDING COMMAND 0x%x\n", c);
  write(fd, writeBuf, writeBufLen);
}

//Trata de iniciar a conexao entre o transmitter e o receiver
int setConnection(char * porta, int user) //0 - writer, 1 -
    receiver
{
  struct termios newtio;

    int sum = 0, speed = 0, fd;

    fd = open(porta, O_RDWR | O_NOCTTY );
    if (fd <0) {perror(porta); return -1; }

    if ( tcgetattr(fd,&oldtio) == -1) { /* save current port
       settings */
      perror("tcgetattr");
      return -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]    = 0;   /* inter-character timer unused
       */
    newtio.c_cc[VMIN]     = 1;   /* blocking read until 5 chars
       received */

    tcflush(fd, TCIOFLUSH);

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

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

  (void)signal(SIGALRM, sendFrame); //instala a rotina que envia
      um sigalrm quando houver time-out, chamando a funcao
     sendFrame

  if(user==0) //writer
  {
    writeBuf[0] = FLAG;
    writeBuf[1] = A;
    writeBuf[2] = C_SET;
    writeBuf[3] = writeBuf[1] ^ writeBuf[2];
    writeBuf[4] = FLAG;
    writeBufLen = 5;
    printf("WRITING SET\n");

    globalFD = fd;
    sendFrame();

    if (readFrame(fd, NULL, NULL) != C_UA)
    {
      printf("\tWRONG SET RESPONSE, QUITING\n");
```

```
      return -1;
    }
    else
      printf("CONNECTION ESTABLISHED!\n");
  }
  else //reader
  {
    printf("AWAITING CONNECTION\n");
    if(readFrame(fd, NULL, NULL) != C_SET)
    {
      printf("\tSET FRAME MALFORMED, QUITING\n");
      return -1;
    }
    else
    {
      sendControlFrame(fd, C_UA);
      printf("CONNECTION ESTABLISHED\n");
    }
  }
  return fd;
}

int readData(int fd, char * buffer)
{
  int res, readC = readFrame(fd, buffer, &res);
  if(readC == C_DISC) //disconnect recebido
  {
    return 0;
  }
  else if(readC == -1) //invalid bcc2
  {
    printf("\nINVALID DATA, SENDING REJECT\n");
    numRejects++;
    sendControlFrame(fd, C_REJ ^ (c<<4)); //passa os 4 bits do c
        para a esquerda
    return -1;
  }
  else
  {
    sendControlFrame(fd, C_RR ^ (c<<4));
    return res;
  }
}

int readFrame(int fd, char * buffer, int *res)
{
  int i = 0, bcc2 = 0, state = 0;
    unsigned char readByte, readC;

    while(1)
    {
    read(fd, readBuf, 1);
    readByte = readBuf[0];
    switch(state)
    {
    case 0: //FLAG
      if(readByte == FLAG)
        state=1;
      break;
    case 1: //A
```

```
    if(readByte == A)
      state=2;
    else if(readByte == READER_A)
      state=5;
    else if(readByte != FLAG)
      state = 0;
    break;
  case 2: //C
    readC = readByte;
    if(readByte == FLAG)
      state = 1;
    else if(readByte == C_SET || readByte == C_UA || readByte
       == C_DISC || readByte == C_RR ^ (alternaC(c)<<4) ||
       readByte == C_REJ ^ (alternaC(c)<<4) || readByte ==
       C_RR ^ (c<<4) || readByte == C_REJ ^ (c<<4) || readByte
        == 0 || readByte == 2)
      state = 3;
    else
      state = 0;
    break;
  case 3: // BCC1
    if(readByte == A ^ readC)
      state = 4;
    else if(readByte == FLAG)
            state = 1;
        else
            state = 0;
    break;
  case 4: //FLAG
    if(readByte == FLAG)
    {
      //reinicia timeouts
      alarm(0);
      conta=0;
      if (readC == c) //DATA FRAME, NOT DUPLICATE
              {
        numTramas++; //incrementa tramas recebidas (reader)
        *res = i-1; //-1 e o bcc
        if(buffer[i-1] == bcc2)
        {
          c=alternaC(c);
        }
        else
        {
          return -1;
        }
              }
      return readC;
    }
    else
    {
      //printf("\t%x - %c\n", readByte, readByte);
      //printf("I: %i\n", i);

      //destuffing
      if(readByte==0x7d) //se caracter for 0x7d, destuff
      {
        read(fd, readBuf, 1); //le o proximo
        readByte=readBuf[0];
        if(readByte == 0x5e)
```

```
            readByte = 0x7e;
          else if(readByte == 0x5d)
            readByte = 0x7d;
        }

        buffer[i] = readByte; //construir chunk (para escrita no
            ficheiro)
        if(i>0)
          bcc2^=buffer[i-1]; //para nao fazer xor do bcc2 com o
              proprio bcc2 (seria sempre 0)
        i++;
      }
      break;
    case 5: //DISC COMMAND FROM READER
      readC = readByte;
      if(readByte == C_DISC)
        state = 3;
      else
        state = 0;
    }
    }
}

//Trata de terminar a conexao
int disconnect(int fd, int user) //0 writer, 1 reader
{
  writeBuf[0]=FLAG;
  writeBuf[1]=A;
  if(user==1) //needs to change address
    writeBuf[1]=READER_A;
  writeBuf[2]=C_DISC;
  writeBuf[3]=writeBuf[1]^writeBuf[2];
  writeBuf[4]=FLAG;
  writeBufLen=5;
  printf("DISCONNECTING\n");

    globalFD = fd;
  sendFrame(); //Envia trama de controlo com C_DISC
  if(user==0)
  {
    int readC = readFrame(fd, NULL, NULL);
    if(readC != C_DISC)
      disconnect(fd, user); //wrong response, send disc again
    else
      sendControlFrame(fd, C_UA);
  }
  else
  {
    if(readFrame(fd, NULL, NULL) != C_UA) //se nao receber UA
    {
      printf("INVALID DISCONNECT RESPONSE\n");
      return -1;
    }
  }

  sleep(1); //wait before closing connection

  //close connection
  if ( tcsetattr(fd,TCSANOW,&oldtio) == -1) {
    perror("tcsetattr");
```

```
    return -1;
  }

  close(fd);
  //IMPRIME ESTATISTICAS
  printf("Numero de tramas I: %i\n", numTramas);
  printf("Numero de timeouts: %i\n", numTimeouts);
  printf("Numero de rejects I: %i\n", numRejects);
  return 0;
}

//Funcao auxiliar que constroi e envia um trama de dados com um
    pacote 'packet'
int sendData(int fd, char *packet, int packetChars)
{
  //cabecalho da trama
  writeBuf[0] = FLAG;
  writeBuf[1] = A;
  writeBuf[2] = c;
  writeBuf[3] = writeBuf[1] ^ writeBuf[2];
  int i, bcc2 = 0;

  //dados
  for (i = 4; i<packetChars + 4; ++i)
  {
    writeBuf[i] = packet[i - 4];
    //printf("writeBuf[%i] -> %x\n", i, writeBuf[i]);
    bcc2 ^= writeBuf[i];
  }
  writeBuf[i++] = bcc2;
  int packetCharsAfterStuffing = stuffBytes(packetChars+1); //
     Numero de caracteres para fazer stuffing
  //printf("%x\n", bcc2);
  i=packetCharsAfterStuffing+4; //posicao no indice seguinte e
     igual ao novo numero de caracteres a transmitir + cabecalho
      da trama
  writeBuf[i++] = FLAG;
  writeBufLen = i;

  printf("WRITING DATA FRAME\n");
  globalFD = fd;
  sendFrame();

  if (readFrame(fd, NULL, NULL) == (C_RR ^ (alternaC(c)<<4)))
     //RR
  {
        c=alternaC(c);
    printf("\tRECEIVER READY!\n");
  }
  else //REJECT OR INVALID RESPONSE
  {
    numRejects++;
    printf("\tREJECT! RESENDING!!!\n");
    sendData(fd, packet, packetChars); //reenvia
  }

  numTramas++; //incrementa tramas enviadas (writer)

  return packetChars;
}
```

```
int alternaC(int c)
{
    if(c==0)
        return 2;
    else
        return 0;
}
```

Read.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>

#define CHUNK_SIZE 100

int n = 0;

unsigned int fileSize;

char fileName[255];
FILE * pFile;

int packetStateMachine(char *buffer);
int llopen(char *porta);
int llclose(int fd);
int llread(int fd, char *buffer);
```

```
#include "read.h"

//Maquina de estados para um trama de dados e respectivo pacote
int packetStateMachine(char *buffer)
{
  unsigned char c = buffer[0];
  switch(c)
  {
    case 0:
    {
      printf("\tDATA PACKET\n");
      if(buffer[1]==n) //se nao for um pacote duplicado, grava
          para ficheiro
      {
          unsigned char l2 = buffer[2], l1 = buffer[3];
          int k = 256*l2+l1;
          fwrite(buffer+4, sizeof(char), k, pFile); //escreve
              para ficheiro
          if(n==255)
              n=0;
          else
              n++;
      }
      break;
    }
    case 1:
    {
      printf("\tSTART PACKET\n");
      unsigned char t1 = buffer[1], l1 = buffer[2], i, tmp = l1;
      for(i=3;i<l1+3;++i)
          fileSize |= (((unsigned char)buffer[i]) << (8*--tmp));

      unsigned char t2 = buffer[i++], l2 = buffer[i++], j=i;
      for(;i<l2+j;i++)
          fileName[i-j]=buffer[i];

      //printf("\t%s\n", fileName);

      break;
    }
    case 2:
    {
      printf("\tEND PACKET\n");
      if(fileSize != ftell(pFile)) //compara tamanho do ficheiro
          do pacote inicial com o realmente escrito para o
          ficheiro
          printf("\nERROR TRANSFERING: WRONG FILE SIZE\n");

      break;
    }
  }
  return 0;
}

int llopen(char *porta)
{
  return setConnection(porta, 1); //reader
}
```

```
int llclose(int fd)
{
  return disconnect(fd, 1); //1 - reader
}

int llread(int fd, char *buffer)
{
  return readData(fd, buffer);
}

int main(int argc, char** argv)
{
  if (argc < 2){
    printf("Usage:\tnserial SerialPort\n\tex: nserial /dev/ttyS1
       \n");
    exit(1);
  }
  //Open file for writing
  pFile = fopen ( "tmp" , "wb" );
  if (pFile==NULL) {fputs ("File error",stderr); exit (1);}

  int fd = llopen(argv[1]);
  if(fd<0)
    return -1;

  char chunkBuf[CHUNK_SIZE+5];

  while(1)
  {
    int res = llread(fd, chunkBuf);
    if(res == 0) //disconnect request received
    {
      if(llclose(fd) < 0)
        return -1;
      break;
    }
    else if(res != -1) //not invalidBCC
      packetStateMachine(chunkBuf);
  }
  printf("\nALL DONE\n");
  //sleep(1);

  fclose(pFile);
    rename("tmp", fileName); //rename file
    return 0;
}
```

```c
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>

#define CHUNK_SIZE 100

int llopen(char *porta);
void sendFile(int fd, unsigned int fileSize, char *fileName,
    char *fileContent);
void sendControlPacket(int fd, int packetC, unsigned int
    fileSize, char *fileName);
void sendDataPacket(int fd, char *packet, int dataChars,
    unsigned char n);
int openFile(char *fileName, char **fileContent);
int llclose(int fd);
int llwrite(int fd, char * buffer, int length);
```

```
#include "write.h"

//Trata de iniciar a conexao entre o transmitter e o receiver
int llopen(char *porta)
{
  return setConnection(porta, 0); //0 - writer
}

//Trata do envio de dados, no devido formato
void sendFile(int fd, unsigned int fileSize, char *fileName,
   char *fileContent)
{
  unsigned int i, n=0;
  //pacote de controlo start
  sendControlPacket(fd, 1, fileSize, fileName);

  char chunkBuf[CHUNK_SIZE+4]; //tamanho do chunk a enviar mais
      cabecalho

  for(i=0;i<(fileSize/CHUNK_SIZE);i++) //whole chunks
  {
    memcpy(chunkBuf+4, &fileContent[i*CHUNK_SIZE], CHUNK_SIZE);
        //ADD 4 to accommodate packet information
    sendDataPacket(fd, chunkBuf, CHUNK_SIZE, n);
    if(n==255)
      n=0;
    else
      n++;
  }

  //last bytes
  if(fileSize%CHUNK_SIZE>0)
  {
    memcpy(chunkBuf+4, &fileContent[i*CHUNK_SIZE], fileSize%
        CHUNK_SIZE); //ADD 4 to accommodate packet information
    sendDataPacket(fd, chunkBuf, fileSize%CHUNK_SIZE, n);
  }

    //pacote de controlo end
  sendControlPacket(fd, 2, fileSize, fileName);
}

//Constroi e envia trama de dados com um pacote de controlo
void sendControlPacket(int fd, int packetC, unsigned int
   fileSize, char *fileName)
{
  char controlPacket[255];
  controlPacket[0]=packetC; //C
  controlPacket[1]=0; //T1 - tamanho do ficheiro
  int i=3, currentByte, significant = 0;

  for(currentByte=sizeof(fileSize);currentByte>0;currentByte--)
     //percorre inteiro byte a byte do mais significativo para o
        menos
  {
    unsigned char byte = (fileSize >> (currentByte-1)*8) & 0xff;
        //passa bytes individuais(octetos) para controlPacket

    if(!significant) //enquanto forem nao significativos
      if(byte!=0) //se byte actual nao for zero
```

```
            significant = 1; //todos os seguintes sao significativos
        if(significant) //para todos os bytes significativos, enviar
          controlPacket[i++]=byte; //V1
    }

    controlPacket[2]=i-3; //L1
    controlPacket[i++] = 1; //T2 - nome do ficheiro
    controlPacket[i++] = strlen(fileName); //L2

    int j;
    for(j=0;j<strlen(fileName);j++) //escreve caracteres
        individuais do nome do ficheiro
      controlPacket[i++] = fileName[j]; //V2

    printf("WRITING CONTROL PACKET\n");
    llwrite(fd, controlPacket, i);
}

//Envia trama de dados com um pacote de dados
void sendDataPacket(int fd, char *packet, int dataChars,
     unsigned char n)
{
    //cabecalho do pacote
    packet[0]=0;
    packet[1]=n;
    packet[2]=dataChars/256;
    packet[3]=dataChars%256;
      printf("WRITING DATA PACKET\n");
    llwrite(fd, packet, dataChars+4); //por causa do cabecalho
}

int llwrite(int fd, char * buffer, int length)
{
    return sendData(fd, buffer, length);
}

//Abre ficheiro para leitura em binario, devolvendo o tamanho em
     bytes do ficheiro
int openFile(char *fileName, char **fileContent)
{
    FILE * pFile;
    long lSize;
    size_t result;

    pFile = fopen ( fileName , "rb" );
    if (pFile==NULL) {fputs ("File error",stderr); exit (1);}

    // obtain file size:
    fseek (pFile , 0 , SEEK_END);
    lSize = ftell (pFile);
    rewind (pFile);

    // allocate memory to contain the whole file:
    *fileContent = (char*) malloc (sizeof(char)*lSize);
    if (*fileContent == NULL) {fputs ("Memory error",stderr); exit
        (2);}

    // copy the file into the buffer:
    result = fread (*fileContent,1,lSize,pFile);
    if (result != lSize) {fputs ("Reading error",stderr); exit (3)
        ;}
```

```
  /* the whole file is now loaded in the memory buffer. */

  // terminate
  fclose (pFile);
  return lSize;
}

int llclose(int fd)
{
  return disconnect(fd, 0); //0 - writer
}

int main(int argc, char** argv)
{
  if (argc < 3){
    printf("Usage:\tnserial SerialPort FileName\n\tex: nserial /
       dev/ttyS1 text.txt\n");
    exit(1);
    }

  unsigned int fileSize;
  char *fileContent, fileName[255];


  strcpy(fileName, argv[2]);

  fileSize = openFile(fileName, &fileContent);

  printf("FILESIZE: %i\n", fileSize);

  int fd = llopen(argv[1]);
  if(fd<0)
    return -1;
  sendFile(fd, fileSize, fileName, fileContent);

  printf("\nALL DONE\n");

  if(llclose(fd) < 0)

    return -1;

  return 0;
}
```