Relatório 1º Trabalho Laboratorial

Protocolo de Ligação de Dados

Mestrado Integrado em Engenharia Informática e Computação

RCOM 2020/2021

Turma 5 Grupo 3:

Bernardo Ramalho - up201704334@fe.up.pt

Ivo Façoco - up201707093@fe.up.pt

Prof. Rui Campos

Prof. Manuel Ricardo

Sumário

No âmbito da disciplina de Redes de Computação foi realizado um projeto com o objetivo de criar uma aplicação capaz de transferir ficheiros entre computadores recorrendo a um protocolo de comunicação não canónica.

O projeto foi concluído com sucesso, uma vez que todos os elementos propostos no enunciado foram implementados. O resultado final foi uma aplicação capaz de estabelecer a ligação entre dois computadores, transferir ficheiros e ao mesmo tempo realizar as verificações necessárias para garantir a integridade dos mesmos.

1. Introdução

O trabalho prático realizado, como foi dito acima, tem como objectivo estabelecer comunicação entre dois computadores, através de uma porta de série, de forma a que seja possível transferir ficheiros entre os computadores.

O relatório serve como base para perceber o conceito por detrás do código desenvolvido, perceber algumas das decisões feitas na implementação de certas funcionalidades e clarificar algum aspecto mais teórico que possa não ser facilmente percebido através do código fornecido.

O relatório está dividido em seções numeradas de 1 a 9:

- Na **secção 1,** fazemos uma breve apresentação do relatório e deve servir como guia para o resto do relatório;
- Na **secção 2,** explicamos a parte mais conceptual do nosso projeto. Ou seja, é nesta secção que percebe a organização base e a forma de utilizar a nossa aplicação;
- Na **seção 3,** mostramos como organizamos o código e explicamos para que servem as funções principais. Recomendamos ler antes de olhar para o código anexado, de forma, a ter uma visão geral de como funciona o código;
- Na **seção 4,** enumeramos os principais casos de uso e mostramos a ordem pela qual as funções são chamadas quando se corre a aplicação;
- Na **secção 5,** explicamos, de uma forma mais aprofundada, o funcionamento das funções, descritas na seção 3, relativas ao protocolo de ligação;
- Na **secção 6,** explicamos, também, de uma forma mais aprofundada, o funcionamento das funções, descritas na seção 3, mas relativas à aplicação;
- Na **secção 7**, enumeramos os testes feitos para verificar que a nossa aplicação tem todas as funcionalidades pedidas no enunciado;
- Na **secção 8**, mostramos os resultados dos nossos testes quanto à eficiência do nosso protocolo.
- Na **secção 9**, falamos sobre as conclusões retiradas a nível do projeto mas também a nível da cadeira para a qual fizemos este projeto.

2. Arquitetura

A aplicação está dividida em duas camadas, *App* e *LinkLayer*, respeitando assim o princípio da independência de camadas. A primeira, tanto no emissor como no recetor, implementa a funcionalidade esperada, sendo esta a transferência de ficheiros entre dois computadores. Para alcançar este objetivo a *App* recorre ao *LinkLayer* (protocolo de ligação

de dados) que contém as funções responsáveis pela configuração das portas de série, envio e receção de ficheiros e verificação da integridade dos dados. Primeiramente, são inicializadas as portas de série em ambos os computadores e de seguida prossegue-se ao envio de dados de um computador para o outro. No caso de envio de dados com sucesso a aplicação termina a ligação estabelecida inicialmente.

A interface com que o utilizador interage é a consola do sistema operativo Linux. Nesta, o utilizador pode especificar o nome do ficheiro a transferir e a porta de série a utilizar. Isto está explicado no nosso README.md, encontrado no código anexado.

3. Estrutura do Código

Para facilitar a organização do código e aumentar a facilidade da leitura do mesmo, criamos, tal como dito no enunciado, duas estruturas: *linkLayer* e *App*. Tal como o nome indica, a estrutura *linkLayer* é utilizada para guardar valores que irão ser usados nos ficheiros relativos à *LinkLayer* enquanto a estrutura *App* é utilizada como mesmo propósito mas nos ficheiros relativos à *App*.

Para representar as tramas utilizamos variáveis a que chamamos buffer, que são char arrays. Estes arrays têm um tamanho máximo especificado com a constante MAX_SIZE. As tramas são guardadas na struct *linkLayer*. Para representar a porta de série é utilizado o file descriptor relativo à mesma.

As funcionalidades da *LinkLayer* estão divididas em 5 funções: *initLinkLayer, llopen*, *llread*, *llwrite* e *llclose*. Todas estas funções são usadas pela *App*.

- A função ***initLinkLayer*** é utilizada para abrir a porta de série indicada e alterar as definições dessa mesma porta. Antes de fazer as alterações, as definições são guardadas na struct *linkLayer,* de forma a poderem ser revertidas no final do programa.
- A função ***llopen*** serve para verificar que a conexão foi aberta, utilizando o file descriptor e uma flag de indicação se é emissor ou receptor enviada pela *App*.
- A função ***llread*** é utilizada pelo receptor para ler os frames que são enviados pela porta de série. É também nesta função que fazemos verificações quanto às flags e possíveis erros nos dados.
- A função ***llwrite*** é utilizada pelo emissor para enviar os frames pela porta de série. Os dados que têm de ser enviados são passados quando a função é chamada pela *App.* Esta função é responsável por criar as flags necessárias de forma a criar uma trama válida com os dados recebidos.
- A função ***llclose*** serve, em contraste com *llopen*, para confirmar que se pode fechar a conexão. Ao fechar a conexão também é reposta as definições da porta de série que foram modificadas quando se chamou a função *initLinkLayer*.

Nos ficheiros relativos à *App*, temos 3 funções principais: a função ***main(),*** a função *receiveDataFrames* e *sendDataFrames*.

- A função ***receiveDataFrames*** é chamada pelo receptor para ler os dados que foram transferidos pelo emissor. Nesta função chamamos a função *llread,* do protocolo de

ligação, até terem sido lidos um número de blocos de dados especificado na struct *App*;

- A função ***sendDataFrames*** é chamada pelo emissor para enviar o ficheiro para o receptor. Nesta função chamamos a função *llwrite,* do protocolo de ligação, até terem sido enviados todos os blocos correspondentes ao ficheiro;
- A função ***main*** é chamada quando se corre o programa na consola. É aqui que chamamos as funções indicadas acima. A função tem comportamento diferente dependendo se é o receptor ou o emissor que a chama. Esta distinção é feita consoante os argumentos que a função recebe. Se só receber 2 argumentos sabemos que o receptor se forem 3 trata-se do emissor. O emissor recebe mais um argumento que o receptor pois temos de dizer ao emissor qual o nome do ficheiro a transferir enquanto que o receptor recebe isso numa trama.

4. Casos de Uso Principais

Para este projeto foi proposta uma aplicação capaz de enviar ficheiros de um computador para outro recorrendo a uma porta porta de série. Esta aplicação tinha como requisitos o estabelecimento da ligação entre dois computadores, o envio de dados por parte do emissor e a recepção desses dados pelo receptor, que os escreve num ficheiro com o mesmo nome daquele que foi enviado pelo emissor. Isto é na seguinte sequência:

1. Caso seja o emissor, é aberto o ficheiro a enviar e guarda-se as suas propriedades na struct *App*. Se for o receptor, apenas mete a flag status da struct *App* a identificar que é o recetor;
2. Inicia-se a conexão utilizando a função *initLinkLayer*;
3. Utiliza-se a função llopen, tanto no lado do emissor como do receptor, para verificar se a ligação foi bem estabelecida;
4. Agora acontecem duas coisas dependendo se estamos do lado do emissor ou receptor:
    a. No caso do emissor, este envia uma control frame, utilizando uma função auxiliar, para avisar que vai começar a transferir informação. Depois envia todos os dados que tem para enviar utilizando a função *sendDataFrames*;
    b. No caso do receptor, ele tenta receber, utilizando uma função auxiliar, a control frame que o emissor enviou a avisar o início da transferência de informação. Depois, utiliza a função *receiveDataFrames* para ler a informação enviada pelo emissor.
5. No fim de enviar tudo ou receber tudo, é fechada a conexão utilizando o *llclose* e fecha-se os ficheiros que foram abertos.

Os ficheiros usados para testar a aplicação foram uma imagem gif de um pinguim, um ficheiro de texto com os caracteres de escape (de forma a testar o byte stuffing) e uma imagem de alta resolução.

5. Protocolo de Ligação Lógica

O protocolo de ligação lógica tem como objetivo o fornecimento de um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio (canal) de transmissão – neste caso, um cabo série. Este protocolo está representado na camada operacional *LinkLayer*.

5.1 Principais Aspetos funcionais

A camada de protocolo de ligação tem como funções:

- O estabelecimento e terminação da ligação estabelecida entre dois computadores através da porta de série;
- A organização de dados e comandos em tramas e o seu envio através da porta de série;
- O controlo erros através flags e campos dedicados (bcc e bcc2) presentes nas tramas, assim como tempo de espera de resposta e respostas negativas por parte do recetor;
- A delimitação das tramas recorrendo ao uso de sequências especiais de 8 bits (flags);
- A garantia da transparência nas tramas de dados através de byte stuffing.

5.2 Estratégia de Implementação

5.2.1 initLinkLayer()

Esta função está encarregue da configuração da porta de série, começando por inicializar os campos necessários para a camada *LinkLayer* na estrutura de dados *linkLayer*. De seguida, a porta de série é aberta e configurada de acordo com os critérios definidos no enunciado. O valor de retorno é o file descriptor da serial port, para que este possa ser usado dentro da camada da aplicação. A configuração termios ***oldtio*** é guardada na estrutura de dados ***linkLayer***, para que o estado da porta de série possa ser reposto no fim do programa.

5.2.2 llopen()

Esta função tem como objetivo estabelecer e verificar a conexão entre os dois computadores. Os argumentos da função são o file descriptor da porta de série e a designação da máquina (emissora ou receptora), uma vez que a função desempenha tarefas diferentes de acordo com a sua classificação.

Inicialmente, do lado do emissor, a função começa por construir a trama de supervisão com o campo de endereço **A_EM** (0x03) , o campo de controlo **SET** (0x03) e calcula o respectivo **BCC** a ser enviado no campo de proteção. De seguida, envia a trama para o recetor através da porta de série e espera pela confirmação. Se o emissor não obtiver nenhuma resposta ao fim de um segundo (tempo de espera máximo), este volta a enviar a trama e espera novamente pela confirmação. Ao fim de três falhas consecutivas (número máximo de tentativas) o emissor termina a execução do programa, o que indica

que a conexão não foi estabelecida. Para que a função termine com sucesso o receptor deve enviar uma trama não numerada com o campo de endereço **A_EM**, campo de controlo **UA** e um campo de proteção válido. O código desta função pode ser visto no **Anexo 1.**

5.2.3 llwrite()

A função *llwrite* é responsável pelo envio de tramas de informação do emissor para o recetor. Esta tem como parâmetros o file descriptor da porta de série, o buffer que contém os dados a serem enviados e o tamanho do bloco de dados.

Em primeiro lugar, a função calcula o **BCC2** do campo de dados recorrendo à função auxiliar ***calcBcc2***, que vai recursivamente realizar a operação XOR entre os bytes de dados e os resultados previamente obtidos. Seguidamente, sucede-se o tratamento dos dados onde é assegurada a transparência do campo de dados através da função auxiliar ***stuffBytes***. Esta função substitui no interior da trama todos os octetos com valores **0x7E** e **0x7D** pelas sequências **0x7D 0x5E** e **0x7D 0x5D**, respetivamente. Depois ocorre a construção da trama com o campo de endereço **A_EM** e campo de controlo a **0x00**, no caso de o número de sequência ser **0** e **0x40** quando é igual a **1**. Finalmente, depois deste pré-processamento, a trama é enviada pela porta e o emissor espera pela resposta do receptor. Se não for obtida nenhuma resposta, a trama é reenviada. Se ao fim de três tentativas não tiver sido recebida uma resposta válida o programa acaba em erro. A função só termina em sucesso se a trama não numerada recebida for válida, isto é, se o campo de controlo for igual a **RR** (trama aceite pelo recetor). Se o campo de controlo for igual a **RJ** (trama rejeitada pelo receptor), o emissor reenvia a informação. O código desta função pode ser visto no **Anexo 3.**

5.2.4 llread()

A função *llread* está encarregue da recepção de tramas de informação enviadas pelo recetor. Os parâmetros são o file descriptor da porta de série e buffer de destino dos dados lidos. Esta função lê um byte de cada vez da trama enviada pelo emissor e valida esse byte na máquina de estados. Primeiro valida os bytes correspondentes à cabeça da trama e de seguida começa a leitura do campo de dados da mesma. Se durante a leitura aparecer um byte que coincida com o octeto de escape (0x7D) a máquina de estados transita para o estado de **DESTUFFING** onde a operação de stuffing descrita acima é revertida. Quando a flag delimitadora **DELIM** (0x7E) é encontrada a leitura acaba e o **BCC2** é calculado recorrendo à função auxiliar ***calcBcc2*** descrita na secção anterior. Se o **BCC2** coincidir com o que foi enviado na trama a função envia um reconhecimento positivo (trama com campo de controlo **RR**) ao emissor; senão, envia um reconhecimento negativo (trama com campo de controlo **RJ**). O código desta função pode ser visto no **Anexo 3.**

5.2.5 llclose()

Esta função tem como objetivo a terminação da ligação entre as duas máquinas através da porta de série. O único parâmetro é o file descriptor da porta.

O emissor começa por enviar a trama de supervisão com o campo de controlo **DISC** (0x09), indicando ao recetor que a ligação vai ser fechada. De seguida, o receptor envia a mesma trama que o emissor, diferindo apenas no campo de endereço que agora é **A_RC** (campo de endereço do emissor). Por último, o emissor recebe a trama do recetor e envia uma trama não numerada (campo de controlo **UA**) ao mesmo.

No fim da função as configurações originais da porta de série, inicialmente guardadas na estrutura de dados *linkLayer*, são restauradas. O código desta função pode ser visto no **Anexo 4.**

6. Protocolo de Aplicação

O protocolo de aplicação implementa as funcionalidades da aplicação projetada recorrendo ao protocolo de ligação para o fazer.

6.1 Estratégia de Implementação

6.1.1 sendControlFrame()

Esta função tem como objetivo a construção e envio de pacotes de controlo do nível de aplicação. O único parâmetro é a flag de controlo que se pretende mandar. Esta flag será **C_START** (0x02), quando o emissor pretende iniciar a transferência de pacotes de dados, ou **C_END** (0x03), quando for para terminar o envio de dados. Cada parâmetro enviado no pacote de dados tem o formato **TLV** (Type, Length, Value), em que **T** indica o tipo de parâmetro, **L** indica o tamanho do parâmetro em octetos e **V** o valor que é envio pelo parâmetro. Na nossa implementação só recorremos a três tipos de parâmetros **T_SIZE** (0x00), para indicar o tamanho do ficheiro, **T_FILENAME** (0x01), para o nome do ficheiro e **T_BLOCK** (0x02), que indica o número de divisões do ficheiro em blocos de tamanho predefinido.

A função começa por construir o pacote de dados com os requisitos descritos acima e, por fim, envia o pacote recorrendo à função ***llwrite*** da camada **LinkLayer**.

6.1.2 receiveControlFrame()

Esta função tem está encarregue de receber o pacote de controlo descrito na secção anterior. Como no início o recetor não tem nenhuma informação sobre o ficheiro que vai receber, este necessita do pacote de controlo, para poder guardar a informação necessária.

6.1.3 sendDataFrames()

O papel desta função é o envio dos pacotes de dados provenientes do ficheiro a ser enviado. O pacote de dados é construído com o campo de controlo **C_DATA** (0x01), para indicar ao receptor que se trata de uma pacote de dados, e campo do número de sequência com **sentBlocks** % 255. Os dados são lidos do ficheiros em blocos de tamanho **BLOCK_SIZE**, e, de seguida, são enviados no campo de dados do pacote de dados utilizando a função ***llwrite***. O processo é repetido até o número de blocos enviados for igual ao número de divisão do ficheiro em blocos **BLOCK_SIZE**.

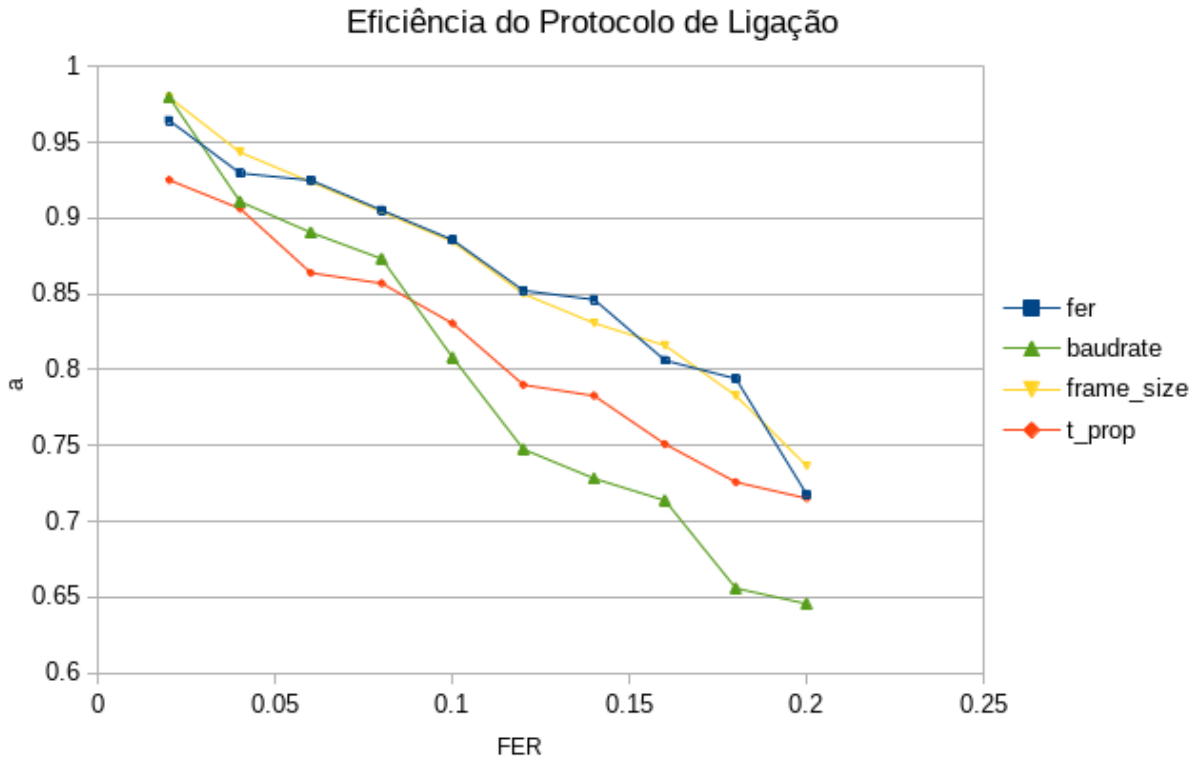
6.1.4 receiveDataFrames()

Esta função está encarregue da recepção dos pacotes de dados enviados pelo emissor através da função ***sendDataFrames***. A cada pacote que recebe ela escreve o campo de dados para um ficheiro com o mesmo nome daquele que está a ser enviado. O processo de leitura é repetido até o número de blocos recebidos for igual ao número de divisão do ficheiro em blocos **BLOCK_SIZE**. Ao longo da execução é feita a verificação do número de sequência para detectar possíveis repetições de pacotes de dados.

7. Validação

Para validar a nossa implementação efetuamos a transferência de vários ficheiros com diferentes tamanhos e formatos. Alguns dos ficheiros foram: o gif do pinguim, que era o objetivo do projeto, um ficheiro de texto com vários octetos de escape, para testar o stuffing e destuffing do protocolo de ligação, e uma imagem png de alta resolução para testar o envio de ficheiros com tamanhos superiores. Todos os ficheiros foram enviados com sucesso.

8. Eficiência do Protocolo de Ligação de Dados



Como era de esperar, a eficiência do protocolo de ligação diminui em todas as situações à medida que o frame error ratio aumenta. Para obter os gráficos efetuamos medições do tempo de propagação para cada pacote recebido e calculamos o S médio para cada frame error rate. Na maior parte dos teste foram utilizados para o tamanho da trama L= 650 bytes; a taxa de dados R =38.4 Mbit/s e um T_f constante de 16.93 µs. As únicas exceções foram nos testes onde era necessária alguma variação destes campos. Os cálculos foram obtidos utilizando as fórmulas do protocolo stop & wait dos slides teóricos sobre as Ligações de Dados.

A linha azul indica os resultados obtidos quando variamos apenas o frame error ratio.

A linha verde indica os resultados obtidos quando variamos o baudrate (capacidade de ligação, em mb/s). Para este teste aumentamos a taxa de dados para 567 Mbit/s. Como podemos ver, a partir do momento em que a probabilidade de haver um erro é maior que 10%, esta variação é a que causa uma queda mais alta na performance. Achamos que isso deve-se ao facto de ao haver mais informação a ser enviada por segundo, existem mais erros a acontecer.

A linha amarela indica os resultados obtidos quando variamos o tamanho de cada frame. Para este teste aumentamos o tamanho para 1000 bytes. Como podemos ver, este aumento não tem um grande impacto na eficiência. Achamos que esse impacto deve-se ao facto de quando um frame tem erro, este é reenviado. Ao termos uma frame maior, a informação demora mais a ser enviada, isto é propagado ainda mais caso haja erro. No entanto, concluímos também que o impacto que vemos não é o verdadeiro impacto pois o tamanho na trama é pequeno em comparação com a capacidade de ligação usada.

A linha laranja indica os resultados obtidos quando incrementamos o tempo de propagação em 5 ms. Como era de esperar houve uma diminuição da eficiência do protocolo em relação à inicial. Isto deve-se ao facto do tempo de envio da trama ser superior o que em caso de erro aumenta substancialmente o tempo de envio da trama.

9. Conclusões

No decorrer deste projeto, fomos implementando todos os elementos propostos nas aulas práticas, o que nos permitiu cumprir todos os objetivos propostos no enunciado. No final do projeto temos uma aplicação capaz de enviar ficheiros entre dois computadores através de uma porta de série e que verifica validade dos dados enquanto os envia, de forma a garantir a integridade dos mesmos.

Após os vários testes, com vários tipos de ficheiros, não encontramos nenhum bug que impedisse o funcionamento da aplicação nem o seu desempenho. Se pudéssemos adicionar alguma coisa a este projeto seria a possibilidade de receber um ficheiro e enviar esse mesmo ficheiro para mais do que um computador.

Em termos pessoais, achamos que este projeto nos enriqueceu e nos fez perceber melhor a matéria relativa à cadeira de RCOM. Permitiu-nos por em práticas aquilo que é lecionado nas aulas teóricas de uma forma interativa e com uma curva de aprendizagem adequada.

Concluindo, achamos este projeto uma mais valia, não só para nós mas para todos os estudantes, pois permitiu o aprofundamento dos conhecimentos lecionados nas aulas teóricas e a consolidação das práticas de comunicação entre duas máquinas na linguagem C.

Anexos

```c
int llopen(int fd, int status) {

    setAlarmFlags();

    switch (status) {
        case EMT_STAT: // Emissor
            if (EmtSetupConnection(fd)) {
                perror("Couldn't setup connection.\n");
                close(fd);
                return -1;
            }
            break;
        case RCV_STAT:  // Receiver
            if (RcvSetupConnection(fd)){
                perror("Couldn't setup connection.\n");
                close(fd);
                return -1;
            }
            break;
        default:
            close(fd);
            perror("Invalid flag.\n");
            return -1;
    }

    return 0;
}
```

Anexo 1 - Código relativo à função llopen.

```c
int llclose(int fd) {

  setAlarmFlags();

  int ret = 0;
  if (ll.status == EMT_STAT) {
    ret = EmtCloseConnection(fd);
  }
  else {
    ret = RcvCloseConnection(fd);
  }

  // Restore the port settings
  if ( tcsetattr(fd,TCSANOW,&ll.oldtio) == -1) {
    perror("tcsetattr");
    return -1;
  }

  return ret;

}
```

Anexo 2 - Código relativo à função llclose.

```c
int llwrite(int fd, unsigned char *data, int size) {

  bool rcvRR = false;
  int numTries = 0;
  setAlarmFlags();
  static bool ns_set = false; // S starts at 0

  bool sentData = false;

  do {

    unsigned char bcc2 = calcBcc2(data, 0, data[0], size-1);

    data[size-1] = bcc2; //add bcc2 to last position

    unsigned char stuffed[MAX_SIZE];
    int ndata = stuffBytes(data, size+1, stuffed);

    // Intialize data frame header
    unsigned char buffer[ndata + 6];
    buffer[0] = DELIM;
    buffer[1] = A_EM;
    buffer[2] = ns_set ? 0x40 : 0x00; // Control flag: S=1 --> 0x40 ; S=0 --> 0x00
    buffer[3] = buffer[1]^buffer[2];

    int buffPos = 4;
    // Concatenating stuffed data buffer
    for (int i = 0; i < ndata; i++) {
        buffer[i+buffPos] = stuffed[i];
    }

    buffPos += ndata;

    buffer[buffPos] = DELIM;


    // Send data
    int nbytes = ndata + 6;
    int n = write(fd, buffer, nbytes);

    alarm(ll.waitTime);

    // Receive acknowledgement from receiver
    if (!receiveFrame(fd, A_EM, ns_set ? (0x0F & RR) : RR)) { // If S=0 expect to receive S=1
        ll.st->sentFrames++;
        ns_set = !ns_set;
        return n; // Success
    }
    else {
      printf("Invalid acknowledment\n");
      numTries++;
    }

  } while (!sentData && (numTries < ll.numTransmissions));

  close(fd);
  return -1;
}
```

Anexo 3 - Código relativo à função llwrite..

```c
int llread(int fd, unsigned char * buffer){

  enum state st = START;

  unsigned char byte;
  int i = 0, res, data_received = 0;
  unsigned char bcc = 0;

  static bool nr_set = 1;

  int index = 0; // Index for return buffer
  while (true) {
    // Read field sent by writenoncanonical
    res = read(fd,&byte,1);
    ll.frame[i] = byte;

    switch (st) {

      case START: // Validate header

        if (ll.frame[i] == DELIM) {
          st = FLAG_RCV;
          i++;
        }
        break;

      case FLAG_RCV:

        if (ll.frame[i] == A_EM) {
          st = A_RCV;
          i++;
        }
        else if (ll.frame[i] == DELIM) {
          continue;
        }
        else {

      case A_RCV:

        if (ll.frame[i] == C_0 || ll.frame[i] == C_1) {
          nr_set = (ll.frame[i] == C_1); // Alternate between 1 and 0
          st = C_RCV;
          i++;
        }
        else if (ll.frame[i] == DELIM) {
            st = FLAG_RCV;
            i = 1;
        }
        else {
            st = START;
            i = 0;
        }
        break;

      case C_RCV:

        bcc = ll.frame[1]^ll.frame[2];

        if (ll.frame[i] == bcc) {
            st = BCC_OK;
            i++;
        }
        else if (ll.frame[i] == DELIM) {
          st = FLAG_RCV;
          i = 1;
        }
        else {
          st = START;
          i = 0;
        }
        break;
```

```
        case BCC_OK:  // Start reading data

          if (ll.frame[i] == DELIM) { // Reached end of frame
            data_received--;

            unsigned char bcc2 = calcBcc2(ll.frame, 4, ll.frame[4], 4 + data_received);

            if(ll.frame[i-1] == bcc2){    // Accepted frame

              if (sendAcknowledgement(fd, A_EM,  nr_set ? (0x0F & RR) : RR) <= 0) { // If S=1 send S=0
                perror("Couldn't send acknowledgement.\n");
                return -1;
              }

              st = START; // For further use
              ll.st->receivedFrames++;
              return i;
            }
            else{   // Rejected frame

              if (sendAcknowledgement(fd, A_EM, nr_set ? (0x0F & RJ) : RJ) <= 0) {
                  perror("Couldn't send acknowledgement.\n");
                  return -1;
                }

              printf("Rejected data packet.\n");

              st = START;
              i = 0;
              data_received = 0;
            }
          }
          else {              // Add byte to frame
            if(ll.frame[i] == 0x7d){
              st = DESTUFFING;
            }
            else{
              buffer[index++] = ll.frame[i];
              data_received++;
              i++;
            }
          }
          break;
        case DESTUFFING:

          if(ll.frame[i] == 0x5e){
            ll.frame[i] = 0x7e;
          }
          else if(ll.frame[i] == 0x5d){
            ll.frame[i] = 0x7d;
          }

          buffer[index++] = ll.frame[i];
          data_received++;
          i++;

          st = BCC_OK;

        break;

        }
      }

      return -1;
}
```

Anexo 4 - Código relativo à função llread.