

REDES DE COMPUTADORES

PROTOCOLO

DE

LIGAÇÃO DE DADOS



Henrique Silva – up202007242

Tiago Branquinho – up202005567

**SUMÁRIO**

Este relatório foi elaborado no âmbito do primeiro trabalho prático da unidade curricular “Redes de Computadores”, pertencente à Licenciatura em Engenharia Informática e Computação. Neste trabalho, foi-nos proposta a transferência de dados usando uma aplicação que recorresse a um protocolo de ligação de dados fiável, mesmo em condições de interrupção e de adição de “ruído” à transmissão. Este trabalho foi implementado em C, no Linux, usando portas série RS-232.

Conclui-se que a proposta foi cumprida com sucesso pois todos os objetivos estabelecidos foram alcançados.

**INTRODUÇÃO**

O objetivo deste trabalho divide-se em duas partes, uma relativa ao protocolo de ligação de dados e outra à aplicação em si. O objetivo do protocolo de ligação de dados é fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um canal de transmissão, neste caso uma porta de série. O objetivo da aplicação é desenvolver um protocolo de aplicação muito simples para transferência de um ficheiro, usando um serviço fiável oferecido pelo protocolo de ligação de dados.

O objetivo do relatório é documentar o código, explicando a nossa implementação em função do guião que nos foi apresentado. O relatório tem a seguinte estrutura:

* Arquitetura – Esclarecimento sobre os blocos funcionais e interfaces.
* Estrutura do código – Explicitação das *APIs*, das principais estruturas de dados e funções, e da sua relação com a arquitetura.
* Casos de uso principais – Identificação destes e das suas sequências de chamada de funções.
* Protocolo de ligação lógica – Identificação dos seus principais aspetos funcionais e descrição da estratégia de implementação dos mesmos, apresentando extratos de código.
* Protocolo de aplicação – Identificação dos seus principais aspetos funcionais e descrição da estratégia de implementação dos mesmos, apresentando extratos de código.
* Validação – Descrição dos testes efetuados, (com apresentação quantificada dos resultados)
* Eficiência do protocolo de ligação de dados – Caracterização da mesma, efetuada recorrendo a estatísticas medidas sobre o código desenvolvido. Comparação desta com a de um *protocolo Stop&Wait*.
* Conclusões – Sintese da informação apresentada e reflexão dos objetivos de aprendizagem alcançados.

**ARQUITETURA**

O trabalho segue uma arquitetura em camadas, já que é possível identificar duas camadas independentes: a da aplicação, e a da ligação de dados.

A camada da ligação de dados é a responsável pelo estabelecimento da ligação, garantindo a integridade do protocolo. Assim, é a responsável pela comunicação direta com a porta de série, gerindo a abertura, fecho, leitura e escrita na mesma. É nesta camada que ocorre a verificação de erros, garantindo-se assim a fiabilidade da transferência. A camada da ligação de dados oferece serviços à camada da aplicação, e, deste modo, é a de mais baixo nível deste projeto.

A camada da aplicação é a responsável pelo envio e receção de ficheiros, estando dividida em dois blocos, o emissor e o recetor. Esta camada acede aos serviços fornecidos pela camada de ligação de dados, invocando funções da mesma. Deste modo, é considerada a de mais alto nível deste projeto.

**ESTRUTURA DO CÓDIGO**

A camada da ligação de dados é representada por uma estrutura de ligação de dados. Esta estrutura contém o dispositivo associado à porta de série, a velocidade de transmissão, o número de tentativas em caso de falha, o valor do temporizador e a entidade que está a usufruir dos seus serviços, recetor ou emissor, que é representada por uma enumeração.

typedef enum

{

LlTx,

LlRx,

} LinkLayerRole;

typedef struct

{

char serialPort[50];

LinkLayerRole role;

int baudRate;

int nRetransmissions;

int timeout;

} LinkLayer;

As principais funções da camada de ligação de dados são llopen(), que abre uma conexão entre o recetor e o emissor, llread(), que permite a leitura de um pacote, llwrite(), que permite o envio de um pacote, llclose(), que fecha a conexão, permitindo a impressão de estatísticas referentes à mesma e, por fim, int set\_up\_port() e int close\_port(), que permitem a abertura e o fecho da porta de série, respetivamente. As duas últimas encontram-se no ficheiro *port\_operations.c*, enquanto as restantes se encontram no ficheiro *link\_layer.c*

int llopen(LinkLayer connectionParameters);

int llwrite(const unsigned char \*buf, int bufSize);

int llread(unsigned char \*packet);

int llclose(int showStatistics);

int set\_up\_port(char \*serialPortName);

int close\_port(int fd);

A camada da aplicação é representada por estrutura de aplicação. Esta estrutura contém o descritor correspondente à porta de série e o estado, recetor ou emissor.

struct applicationLayer {

int fileDescriptor;

int status;

};

As principais funções da camada da aplicação são receiveFile(), sendFile(), buildDataPacket(), parseDataPacket(), buildControlPacket(), parseControlPacket(), e applicationLayer(), a função principal. Esta última, para além de preencher a estrutura LinkLayer, vai invocar, de acordo com o *status*, sendFile(), que permite enviar um ficheiro através da porta de série, ou receiveFile(), que permite a receção de um ficheiro através da porta de série. As funções de *parse* são invocadas pelo recetor, enquanto as de *build* são invocadas pelo emissor.

int buildDataPacket(unsigned char \*packet, int sequenceNumber, unsigned char \*data, int dataLength);

int parseDataPacket(unsigned char \*packet, unsigned char \*data, int \*sequenceNumber);

int buildControlPacket(unsigned char \*packet, unsigned char control, int fileSize, const char \*fileName);

int parseControlPacket(unsigned char \*packet, int \*fileSize, unsigned char \*fileName);

int sendFile(const char \*filename, char \*serialPort);

int receiveFile(char \*filename, char \*serialPort);

void applicationLayer(const char \*serialPort, const char \*role, int baudRate,

int nTries, int timeout, const char \*filename);

**CASOS DE USO PRINCIPAIS**

O utilizador pode escolher as configurações, o nome do ficheiro a ser enviado e o nome com que se deverá criar no recetor através da consola. Em alternativa pode recorrer ao *Makefile* existente, iniciando o sistema na configuração padrão.

No caso do emissor, o sistema irá invocar a função sendFile(), que, primeiramente, tenta abrir o ficheiro a ser lido. Em seguida, é invocada a função llopen(), de forma a estabelecer uma conexão entre o emissor e o recetor, através do envio de uma trama SET e receção de uma trama UA. Em seguida, é invocada a função buildControlPacket(), que cria um *control packet*, que contém a indicação de inicio de transmissão. Esse *packet* é posteriormente enviado para o recetor, através a função llwrite(). Após este início, é implementado um ciclo que vai lendo porções do ficheiro, criando *data packets* que as contêm, e enviando esses *packets*, recorrendo à função llwrite(). Seguidamente, envia um *control packet* criado pela função buildControlPacket() que sinaliza o final da transmissão. Por fim é invocada a função llclose(), de forma a fechar a conexão com o recetor.

No caso do recetor, o sistema irá invocar a função receiveFile(), que invoca a função llopen(), desta vez recebendo a trama SET e enviando a trama UA. Em seguida, é invocada a função llread() de forma a receber o *control packet* enviado pelo emissor, que posteriormente é processado, através da função parseControlPacket(). Depois disso, o ficheiro a ser criado é aberto, e é implementado um ciclo que vai recebendo e processando *data packets* através da função llread() e parseDataPacket(), respetivamente e, por fim, adicionando a informação presente nesses *packets* ao novo ficheiro criado. Seguidamente, é invocada a função llread() de forma a receber o *control packet* enviado pelo emissor, que posteriormente é processado através da função parseControlPacket(). Por fim, é invocada a função llclose(), de forma a fechar a conexão com o emissor.

**PROTOCOLO DE LIGAÇÃO LÓGICA**

No protocolo de ligação lógica, foram implementadas quatro funções: llopen(), llwrite(), llread() e llclose(). Estas funções servem como interface para o protocolo de aplicação utilizar as funcionalidades do protocolo de ligação lógica, e lidam com as quatro fases da transferência de dados. Começando pelo llopen(), é estabelecida a ligação entre o emissor e o recetor, seguem-se llwrite(), onde é enviada uma trama de informação até ser recebida uma resposta de confirmação, e llread(), onde é recebida uma trama de informação e enviada uma resposta de confirmação, e por fim llclose(), onde é finalizada a ligação entre o emissor e o recetor. As leituras de cada trama são feitas através de uma máquina de estados, que as recebe byte a byte, mudando o estado consoante o byte recebido, de modo a que apenas se chega ao estado final se a trama recebida tiver um formato válido.

LLOPEN()

Esta função começa por estabelecer ligação com a porta de série indicada, armazenar as informações enviadas pelo protocolo de aplicação e instalar o *alarm handler*.

serialPort = connectionParameters.serialPort;

    role = connectionParameters.role;

    baudRate = connectionParameters.baudRate;

    nRetransmissions = connectionParameters.nRetransmissions;

    timeout = connectionParameters.timeout;

    if((fd = set\_up\_port(serialPort)) == -1){

        printf("Couldn't open port communication");

        return -1;

    }

    alarmHandlerInstaller();

Após realizadas as operações iniciais, esta função procederá de formas diferentes consoante o parâmetro *role*.

Caso se trate do emissor será enviadauma trama SET e ficará a aguardar por uma resposta UA para confirmação.

if(connectionParameters.role == LlTx){

        unsigned char responseBuffer[BUF\_SIZE\_SF];

        if (createSFrame(frame, FIELD\_A\_T\_INIT, SET) != 0){

            close\_port(fd);

            return -1;

        }

        if (sendFrame(frame, fd, frameLength) == -1){

            close\_port(fd);

            return -1;

        }

        printf("SET sent\n");

        stop = FALSE;

        int bytesRead = -1;

        currentRetransmission = 0;

        relay = FALSE;

        alarm(timeout);

        unsigned char controlByte[1] = {UA};

        while(!stop){

            bytesRead = readSFrame(responseBuffer, fd, controlByte, 1, FIELD\_A\_T\_INIT);

            if(relay){

                sendFrame(frame, fd, frameLength);

                relay = FALSE;

            }

            if(bytesRead >= 0){

                alarm(0);

                stop = TRUE;

            }

        }

        if(bytesRead == -1){

            printf("Couldn't read UA. Closing file\n");

            close\_port(fd);

            return -1;

        }

        printf("UA received\n");

        return 1;

    }

Caso se trate do recetor, este aguardará por receber uma trama SET enviando por sua vez uma resposta UA após a sua receção para confirmação.

if(connectionParameters.role == LlRx){

        unsigned char controlByte[1] = {SET};

        if (readSFrame(frame, fd, controlByte, 1, FIELD\_A\_T\_INIT) != 0){

            close\_port(fd);

            return -1;

        }

        printf("SET received\n");

        if(createSFrame(frame, FIELD\_A\_T\_INIT, UA) != 0){

            close\_port(fd);

            return -1;

        }

        if(sendFrame(frame, fd, frameLength) < 0){

            close\_port(fd);

            return -1;

        }

        printf("UA sent\n");

        gettimeofday(&start, NULL);

        return 1;

    }

LLWRITE()

Esta função será apenas chamada pelo emissor para enviar tramas e é composta pelos seguintes passos:

* Criação de uma trama de informação utilizando a função createIFrame();

if(createIFrame(frame, controlByte, buf, bufSize) != 0){

        close\_port(fd);

        return -1;

    }

* Realização do *stuffing* da trama utilizando a função stuffIFrame();

 if((length = stuffIFrame(frame, frameSize)) < 0){

        close\_port(fd);

        return -1;

    }

* Envio da trama utilizando a função sendFrame();

 if((bytesWritten = sendFrame(frame, fd, frameLength)) == -1){

            close\_port(fd);

            return -1;

        }

* Receção de uma resposta RR ou REJ utilizando a função readSFrame();

bytesRead = readSFrame(responseBuf, fd, controlBytes, 2, FIELD\_A\_T\_INIT);

Caso se trate de uma resposta RR o programa sairá da função, e pelo contrário, caso se trate de uma resposta REJ será repetido o processo até receber uma resposta positiva.

Se nunca for recebida uma resposta positiva irá eventualmente ocorrer um *timeout* após um número definido de tentativas, desencadeado pelo *alarm handler* que manipula as variáveis globais que controlam o loop, *relay* e *stop*.

while(!stop){

            bytesRead = readSFrame(responseBuf, fd, controlBytes, 2, FIELD\_A\_T\_INIT);

            if(relay){

                if(sendFrame(frame, fd, frameLength) < 0){

                    printf("Troubles resending. Closing file\n");

                    close\_port(fd);

                    return -1;

                }

                relay = FALSE;

            }

            if (bytesRead >= 0){

                alarm(0);

                stop = TRUE;

            }

        }

LLREAD()

Esta função será apenas chamada pelo recetor para enviar tramas e é composta pelos seguintes passos:

* Leitura de uma trama de informação utilizando a função readIFrame();

bytesRead = readIFrame(frame, fd, controlBytes, 2, FIELD\_A\_T\_INIT);

* Realização do *unstuffing* da trama utilizando a função unstuffIFrame();

if((packetSize = unstuffIFrame(frame, bytesRead)) < 0){

            printf("Troubles unstuffing. Closing file\n");

            close\_port(fd);

            return -1;

        }

* Verificação do BCC2 e do *sequence number*, e determinação da resposta consoante estes (RR caso o BCC2 esteja correto independentemente do *sequence number* ou caso o BCC2 esteja incorreto e se trate de uma trama duplicada, REJ caso o BCC2 esteja incorreto e se trate de uma nova trama);

if (frame[packetSize - 2] == dataBCC(&frame[4], packetSize - 6)){

            if(controlByte != seqNumber){

                if(controlByte == 0){

                    seqNumber = 1;

                    responseByte = RR1;

                }

                else{

                    seqNumber = 0;

                    responseByte = RR0;

                }

            }

            else{

                for (int i = 0; i < packetSize - 6; i++){

                    packet[i] = frame[4 + i];

                }

                bufferFull = TRUE;

                if(controlByte == 0){

                    seqNumber = 1;

                    responseByte = RR1;

                }

                else{

                    seqNumber = 0;

                    responseByte = RR0;

                }

            }

        }

        else{

            if(controlByte != seqNumber){

                if(controlByte == 0){

                    seqNumber = 1;

                    responseByte = RR1;

                }

                else{

                    seqNumber = 0;

                    responseByte = RR0;

                }

            }

            else{

                if(controlByte == 0){

                    seqNumber = 0;

                    responseByte = REJ0;

                }

                else{

                    seqNumber = 1;

                    responseByte = REJ1;

                }

            }

        }

* Criação de uma trama de supervisão, tendo em conta a resposta adequada, utilizando a função createSFrame();

if((createSFrame(frame, FIELD\_A\_T\_INIT, responseByte)) != 0){

            printf("Troubles sending response. Closing file\n");

            close\_port(fd);

            return -1;        }

* Envio da trama utilizando a função sendFrame();

A função retorna o número de bytes recebidos nessa trama.

LLCLOSE()

Esta função procederá de forma diferente dependendo do parâmetro *role*.

Caso se trate do emissor, será enviada uma trama de supervisão DISC, ficando a aguardar por outra trama de supervisão DISC enviada pelo recetor. Aquando da receção deste será enviada uma resposta UA para confirmação.

if(role == LlTx){

        if((createSFrame(frame, FIELD\_A\_T\_INIT, DISC)) != 0){

            printf("Troubles sending command. Closing file\n");

            close\_port(fd);

            return -1;

        }

        frameLength = BUF\_SIZE\_SF;

        if(sendFrame(frame, fd, frameLength) < 0){

            printf("Troubles sending command. Closing file\n");

            close\_port(fd);

            return -1;

        }

        printf("DISC sent\n");

        stop = FALSE;

        int bytesRead = -1;

        currentRetransmission = 0;

        relay = FALSE;

        alarm(timeout);

        unsigned char controlByte[1] = {DISC};

        while(!stop){

            if((bytesRead = readSFrame(frame, fd, controlByte, 1, FIELD\_A\_R\_INIT)) < 0){

                printf("Error reading DISC\n");

                return -1;

            }

            if(relay){

                sendFrame(frame, fd, frameLength);

                relay = FALSE;

            }

            if(bytesRead >= 0){

                alarm(0);

                stop = TRUE;

            }

        }

        if(bytesRead < 0){

            printf("Troubles reading command. Closing file\n");

            close\_port(fd);

            return -1;

        }

        printf("DISC received\n");

        if((createSFrame(frame, FIELD\_A\_R\_INIT, UA)) != 0){

            printf("Troubles sending response. Closing file\n");

            close\_port(fd);

            return -1;

        }

        if(sendFrame(frame, fd, frameLength) < 0){

            printf("Troubles sending response. Closing file\n");

            close\_port(fd);

            return -1;

        }

        printf("UA sent\n");

    }

Caso se trate do recetor, este aguardará por receber uma trama de supervisão DISC, enviando por sua vez outra trama de supervisão DISC após a sua receção, ficando a aguardar por uma resposta UA para confirmação.

else if(role == LlRx){

        unsigned char controlByteCommand[1] = {DISC};

        if (readSFrame(frame, fd, controlByteCommand, 1, FIELD\_A\_T\_INIT) != 0){

            printf("Troubles receiving command. Closing file\n");

            close\_port(fd);

            return -1;

        }

        printf("DISC received\n");

        if(createSFrame(frame, FIELD\_A\_R\_INIT, DISC) != 0){

            printf("Troubles sending command. Closing file\n");

            close\_port(fd);

            return -1;

        }

        if(sendFrame(frame, fd, frameLength) < 0){

            printf("Troubles sending command. Closing file\n");

            close\_port(fd);

            return -1;

        }

        printf("DISC sent\n");

        stop = FALSE;

        int bytesRead = -1;

        currentRetransmission = 0;

        relay = FALSE;

        frameLength = BUF\_SIZE\_SF;

        alarm(timeout);

        unsigned char controlByteResponse[1] = {UA};

        while(!stop){

            if((bytesRead = readSFrame(frame, fd, controlByteResponse, 1, FIELD\_A\_R\_INIT)) < 0){

                printf("Error reading UA\n");

                return -1;

            }

            if(relay){

                sendFrame(frame, fd, frameLength);

                relay = FALSE;

            }

            if(bytesRead >= 0){

                alarm(0);

                stop = TRUE;

            }

        }

        if(bytesRead < 0){

            printf("Troubles reading response. Closing file\n");

            close\_port(fd);

            return -1;

        }

        printf("UA received\n");

No caso do recetor serão ainda impressas as estatísticas referentes à transferências de dados.

gettimeofday(&end, NULL);

        double time\_spent = ((end.tv\_sec - start.tv\_sec) \* 1e6 + (end.tv\_usec - start.tv\_usec)) \* 1e-6;

        if(showStatistics == TRUE){

            printf("Bits Received =  %d\n", bitsReceived);

            printf("Time spent =  %f\n", time\_spent);

            double R =  bitsReceived/time\_spent;

            double S = R / baudRate;

            printf("Debit received = %lf\n", R);

            printf("Efficiency S = %lf\n", S);

        }

    }

**PROTOCOLO DE APLICAÇÃO**

No protocolo de aplicação, foram implementadas sete funções: buildDataPacket(), parseDataPacket(), buildControlPacket(), parseControlPacket(), sendFile(), receiveFile() e applicationLayer(). A função applicationLayer() será responsável pela distinção entre o emissor e recetor, chamando as funções sendFile() ou receiveFile(), respetivamente. Na função sendFile() serão chamadas as funções de construção de *packets*, buildDataPacket() e buildControlPacket(), e na função receiveFile() serão chamadas as funções de *parsing* de *packets*, parseDataPacket() e parseControlPacket().

APPLICATIONLAYER()

Esta função será responsável por armazenar os dados que mais tarde serão passados ao protocolo de ligação de dados e por distinguir o emissor do recetor.

strcpy(linklayer.serialPort, serialPort);

    if(strcmp(role, "tx") == 0)

        linklayer.role = LlTx;

    else if(strcmp(role, "rx") == 0)

        linklayer.role = LlRx;

    else{

        perror("Role not defined\n");

    }

    linklayer.baudRate = baudRate;

    linklayer.nRetransmissions = nTries;

    linklayer.timeout = timeout;

    if (linklayer.role == LlTx)

    {

        if((sendFile(filename, serialPort) != 0)){

            printf("Error sending file\n");

        }

    }

    else if(linklayer.role == LlRx)

    {

        if((receiveFile(filename, serialPort) != 0)){

            printf("Error receiving file\n");

        }

    }

    else{

        perror("Role not defined");

    }

SENDFILE()

Esta função será apenas chamada pelo emissor e é composta pelos seguintes passos:

* Abrir o ficheiro que será enviado utilizando a função openFile();

FILE \*file = openFile(filename, "r");

    if (file == NULL)

    {

        printf("ERROR OPENING FILE!\n");

        return 1;

    }

* Estabelecer ligação entre portas de série utilizando a função llopen();

if (llopen(linklayer) == -1)

    {

        return 1;

    }

* Construir o *control packet* inicial utilizando a função buildControlPacket();

int fileSize = getFileSize(file);

    unsigned char cSPacket[MAX\_PACK\_SIZE];

    int packetSize = buildControlPacket(cSPacket, START\_TRANSFER, fileSize, filename);

* Enviar o *control packet* utilizando a função llwrite();

if (llwrite(cSPacket, packetSize) == -1)

    {

        if (closeFile(file))

        {

            return 1;

        }

        return 1;

    }

* Criar e enviar tramas enquanto houver informação no ficheiro;

while (!feof(file))

    {

        bytesRead = readBytesFromFile(file, data);

        if (bytesRead != MAX\_DATA\_SIZE && !feof(file))

        {

            printf("ERROR READING\n");

            if (closeFile(file))

            {

                return 1;

            }

            return 1;

        }

        packetSize = buildDataPacket(dPacket, getSequenceNumber(n), data, bytesRead);

        n++;

        if (llwrite(dPacket, packetSize) == -1)

        {

            if (closeFile(file))

            {

                return 1;

            }

            return 1;

        }

    }

* Construir e enviar o *control packet* final;

unsigned char cEPacket[MAX\_PACK\_SIZE];

    packetSize = buildControlPacket(cEPacket, END\_TRANSFER, fileSize, filename);

    if (llwrite(cEPacket, packetSize) == -1)

    {

        return 1;

    }

* Fechar a ligação entre portas de série utilizando a função llclose();

if (llclose(1) == -1)

    {

        return 1;

    }

* Fechar o ficheiro enviado utilizando a função closeFile();

if (closeFile(file))

    {

        return 1;

    }

    return 0;

A função retornará 0 caso a operação tenha sido bem-sucedida e 1 caso contrário.

RECEIVEFILE()

Esta função será apenas chamada pelo recetor e é composta pelos seguintes passos:

* Estabelecer ligação entre portas de série utilizando a função llopen();

if(llopen(linklayer) == -1){

        return 1;

    }

* Ler o *control packet* inicial utilizando a função llread();

if((packetSize = llread(cPacket)) < 0){

        printf("Error reading control packet");

        return 1;

    }

* Dar *parse* ao *control packet* utilizando a função parseControlPacket();

if(parseControlPacket(cPacket, &fileSize, packetFilename) != 0 || cPacket[0] != START\_TRANSFER){

        printf("Error parsing control packet\n");

        return 1;

    }

* Abrir o ficheiro de destino utilizando openFile();

FILE \*file = openFile(filename, "w");

    if(file == NULL){

        return 1;

    }

* Ler e dar *parse* a cada trama recebida até ao *control packet* final, inclusive;

int n = 0;

    int sequenceNumber;

    unsigned char dPacket[MAX\_PACK\_SIZE];

    unsigned char data[MAX\_DATA\_SIZE];

    do

    {

        packetSize = llread(dPacket);

        if(packetSize < 0){

            closeFile(file);

            return 1;

        }

        if(dPacket[0] == CTRL\_DATA){

            if(parseDataPacket(dPacket, data, &sequenceNumber)){

                closeFile(file);

                return 1;

            }

            if(sequenceNumber != getSequenceNumber(n)){

                printf("Different sequence numbers\n");

                closeFile(file);

                return 1;

            }

            n++;

        }

        if(dPacket[0] != CTRL\_END){

            if((writeBytesToFile(file, data, packetSize - 4) != packetSize - 4)){

                closeFile(file);

                return 1;

            }

        }

    } while (dPacket[0] != CTRL\_END);

* Fechar o ficheiro de destino utilizando a função closeFile();

closeFile(file);

* Dar *parse* ao *control packet* final utilizando a função parseControlPacket();

int newFileSize = 0;

    unsigned char newFileName[255];

    if((parseControlPacket(dPacket, &newFileSize, newFileName) != 0) || dPacket[0] != END\_TRANSFER)

    {

        printf("Error parsing control packet\n");

        return 1;

    }

* Verificar se o ficheiro original e o ficheiro de destino são idênticos;

if((fileSize != newFileSize) || (strcmp(newFileName, packetFilename) != 0)){

        printf("Files aren't the same\n");

        return 1;

    }

* Fechar a ligação entre as portas de série utilizando a função llclose();

if(llclose(1) == -1){

        printf("Error closing file\n");

        return 1;

    }

A função retornará 0 caso a operação tenha sido bem-sucedida e 1 caso contrário.

**VALIDAÇÃO**

TESTES REALIZADOS

De forma a verificar a integridade do programa foram efetuados os seguintes testes:

* Envio de um ficheiro sem interrupções
* Envio de um ficheiro com a ligação interrompida em vários momentos da transferência
* Envio de um ficheiro numa ligação com ruído em vários momentos da transferência
* Envio de um ficheiro com diferentes tamanhos para as tramas de informação

RESULTADOS

Todos os testes realizados foram concluídos com sucesso.

**EFICIÊNCIA DO PROTOCOLO DE LIGAÇÃO DE DADOS**

Para testar a eficiência do protocolo de dados foram desenhados testes que foram executados mais do que uma vez para melhor precisão estatística. O ficheiro utilizado para o efeito de teste foi penguin.gif e o tamanho máximo de pacote informação definido foi de 1024B.

VARIAÇÃO DO FER

Variando a percentagem de erros nos blocos BCC1 e BCC2 foram detetados resultados distintos. Isto pode ser explicado pelo facto de que, quando existe um erro no BCC1 irá ocorrer um *timeout* após 3 segundos, enquanto no BCC2 a trama será imediatamente reenviada. Assim, podemos verificar que nos dois casos o aumento da percentagem de erros na trama implica diminuições significativas na eficiência, sendo estas mais acentuadas no caso de erros no BCC1.

VARIAÇÃO DO TEMPO DE PROPAGAÇÃO

Variando o tempo de propagação de cada trama, utilizando a função usleep(), foi possível concluir que quanto maior o tempo de propagação menor a eficiência.

VARIAÇÃO DA CAPACIDADE DE TRANSFERÊNCIA

Variando a capacidade de transferência do protocolo, ao alterar a macro BAUDRATE, pudemos observar que quando esta aumenta a eficiência diminui.

VARIAÇÃO DO TAMANHO DOS PACOTES DE INFORMAÇÃO

Variando o tamanho dos pacotes de informação, ao alterar a macro MAX\_DATA\_SIZE, pudemos constatar que quando esta aumenta a eficiência também aumenta.

**CONCLUSÕES**

A realização deste projeto permitiu-nos obter uma melhor compreensão de um protocolo de transferência de dados, nomeadamente ao nível da máquina de estados, construção de tramas e independência entre camadas.

Concluímos também que o trabalho realizado cumpriu todas as expectativas ao superar todos os testes e ao distinguir corretamente a camada de ligação de dados da camada de aplicação.