U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

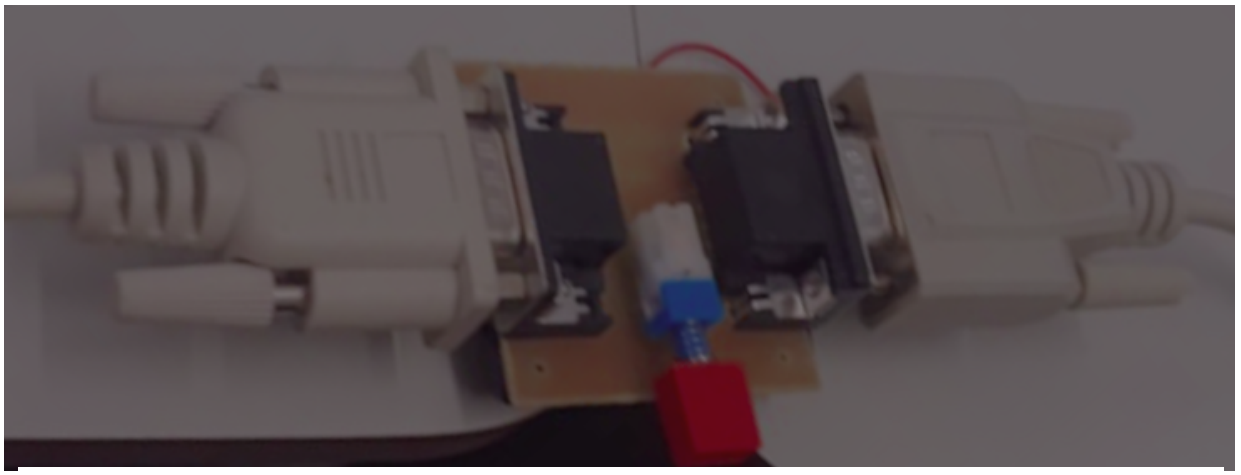
Faculdade de Engenharia da Universidade do Porto

Protocolo de Ligação de Dados

1º Trabalho laboratorial - Redes de Computadores

(2022/2023)

3LEIC05G04



Professor: *Filipe Borges Teixeira*

Trabalho realizado por:

Joana Inês Gonçalves dos Santos up202006279@edu.fe.up.pt

Matilde Pinho Borges Sequeira up202007623@edu.fe.up.pt

Índice

Sumário

Este relatório, realizado no âmbito da unidade curricular Redes de Computadores, expõe o resultado do primeiro trabalho laboratorial, trabalho este cujo objetivo era implementar um protocolo de ligação de dados entre duas máquinas. O projeto foi concluído com sucesso uma vez que foram implementados todos os pontos propostos.

Introdução

O trabalho laboratorial divide-se em dois grandes objetivos, a criação do ***Protocolo de Ligação de Dados*** *(Link Layer)* e da ***Aplicação*** *(Aplication Layer).*
O ***Protocolo de Ligação de Dados*** tem como objetivo fornecer um serviço de comunicação de dados, fiável, entre dois sistemas ligados por um meio (canal) de transmissão – neste caso, um cabo série.
A ***Aplicação*** pretende desenvolver um protocolo de aplicação muito simples para transferência de um ficheiro.
Este relatório tem como objetivo explicar e demonstrar a maneira como realizámos este projeto e como decidimos resolver os diferentes problemas com que nos deparámos.

Arquitetura

Existem dois blocos funcionais, o **writer** (emissor) e o **reader** (recetor). Embora estejam ambos presentes tanto no Application Layer como no Link Layer, o papel de cada um é bastante diferente e têm códigos de execução distintos.
Por exemplo, as funções ***llopen()*** e ***llclose()***, são chamadas tanto pelo Application Layer como pelo Link Layer, no entanto o seu resultado depende de por quem está a ser chamado.

Estrutura do código

1. Link_layer.c

a. Estruturas de dados utilizada

```
typedef enum {
    START,
    FLAG_RCV,
    A_RCV,
    C_RCV,
    BCC_OK
} StateEnum;
```

b. Funções

- ***llopen() -*** Esta função é utilizada tanto pelo bloco funcional **writer(emissor)**, como pelo **reader(recetor)**. No caso do **writer**, nesta função, ele envia a trama de supervisão SET e recebe a trama de supervisão UA, caso tudo tenha corrido bem. Já o **reader**, apenas recebe a trama de supervisão SET e envia a trama UA.

- ***llread()*** - Esta função apenas é acedida pelo **reader (recetor)**. Recebe tramas de informação ***I*** e efetua *byte destuffing.* Caso tudo tenha corrido bem, responde com uma trama RR e envia a informação recebida para a **ApplicationLayer**, caso contrário responde com uma trama REJ para pedir uma nova retransmissão.

- ***llwrite()*** - Esta função apenas é acedida pelo **writer** (emissor), e envia a trama de informação ***I*** após realizar byte stuffing dos dados a enviar. Recebe uma resposta, resposta esta que pode ser a trama de supervisão RR ou REJ. No caso da resposta obtida ser a trama RR, é sinal de que correu tudo bem e a função termina. No caso da resposta ser uma trama do tipo REJ, a informação é reenviada.

- ***llclose()*** - Esta função é utilizada tanto pelo bloco funcional **writer(emissor)**, como pelo **reader(recetor)**. No caso do **writer**, este envia a trama de supervisão DISC, recebe a trama DISC e envia uma trama UA. No caso do **reader**, este recebe a trama de supervisão DISC e envia de novo a trama DISC, por fim recebe a trama de resposta UA.

2. Application_layer.c

a. Estruturas de dados utilizada

```
typedef enum
{
    LlTx,
    LlRx,
} LinkLayerRole;
typedef struct
{
    const char *serialPort;
    LinkLayerRole role;
    int baudRate;
    int nRetransmissions;
    int timeout;
} LinkLayer;
#define MAX_PAYLOAD_SIZE 1000
```

b. Funções:

- ***applicationLayer()*** - É a única função desta secção, e é utilizada, tanto pelo writer (emissor), como pelo reader (recetor).
    - **writer:** Este bloco, abre o ficheiro e cria o pacote de controlo que sinaliza o início da transmissão (START). Enquanto o ficheiro tiver informação por ler, vai ser criado um pacote de dados que é enviado para o recetor, através da chamada da função *llwrite()*. No final é criado o pacote de controlo que sinaliza o fim da transmissão (END).
    - **reader:** Este bloco abre o ficheiro onde a informação vai ser escrita, recebe o pacote de controlo que sinaliza o início da transmissão e, enquanto não receber o pacote que sinaliza o final da transmissão, vai lendo os pacotes de dados e escrevendo a informação recebida no ficheiro.

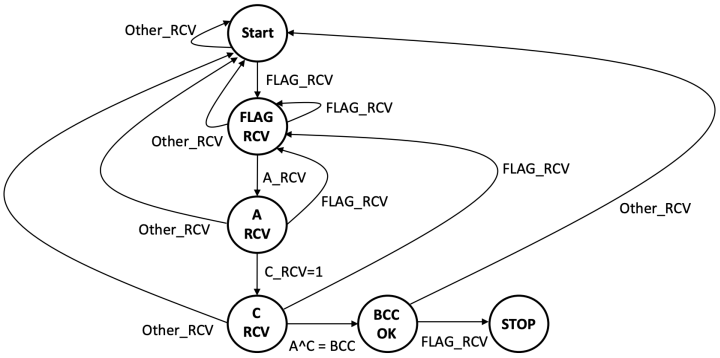
Casos de uso principais

Envio de ficheiros entre dois blocos, um transmissor e um recetor, ligados por um meio de transmissão. Utilizamos uma interface para que seja mais fácil para o utilizador escolher qual o ficheiro a enviar e para que porta de série, e também para definir o ficheiro onde a informação será escrita.

Quando o transmissor escolhe o ficheiro a ser enviado, é configurada e estabelecida a ligação entre os dois blocos. De seguida, o transmissor envia os dados trama a trama e o receptor, à medida que recebe as tramas, escreve no ficheiro de destino a informação. No final, quando já não há mais nada a transferir, a ligação é desconectada.

Protocolo de ligação lógica

No protocolo de ligação lógica foram implementadas as funções *llopen()*, *llwrite()*, *llread()* e *llclose()*, que servem de interface entre a aplicação e as funcionalidades do **data link**. A leitura de qualquer trama é feita a partir de uma máquina de estados, que recebe a informação byte a byte e vai mudando o estado da máquina. O envio de qualquer trama é feito na sua íntegra.



Exemplo da máquina de estados para a receção de uma mensagem SET

1. llopen()

Esta função recebe do **ApplicationLayer** uma struck **LinkLayer** já com os parâmetros preenchidos. De seguida, abre-se a ligação à porta de série.

No caso do transmissor, é enviada a trama de supervisão SET e de seguida inicializado o alarme. Após receber a trama se supervisão UA, este vai lê-la com a ajuda de uma máquina de estados, caso a leitura tenha sido bem sucedida, a função termina. Caso tenha ocorrido algum erro na leitura, é gerado um alarme e o UA é enviado novamente, ou, se o número de retransmissões for ultrapassado, o programa termina.

No caso do receptor, este recebe a trama SET e lê-a recorrendo à máquina de estados. No final envia a trama de resposta UA.

2. llread()

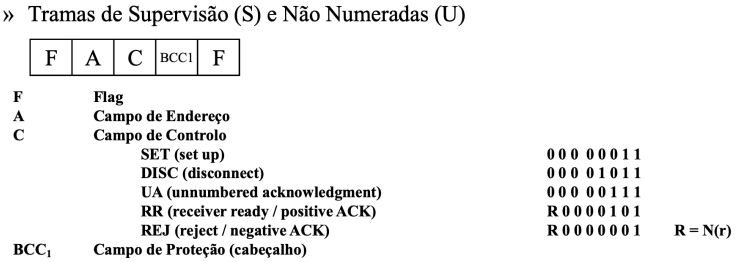
Esta função recebe um buffer que é usado para enviar a informação recebida para o **ApplicationLayer**. Em primeiro lugar, através de uma máquina de estados, lêmos byte a byte a trama de informação ***I***, de acordo com a seguinte divisão:

F	A	C	BCC1	D_1	Dados	D_N	BCC2	F

F	**Flag**
A	**Campo de Endereço**
C	**Campo de Controlo**
D_1 ... D_N	**Campo de Informação (contém pacote gerado pela Aplicação)**
$BCC_{1, 2}$	**Campos de Protecção independentes (1 – cabeçalho, 2 – dados)**

Se nada correu mal no cabeçalho da trama (BCC1 válido), começamos a ler os bytes de dados e a fazer destuffing. Em seguida, é feita a verificação do BCC2. Verificação

esta feita através da comparação do bcc2 recebido com o calculado do xor dos dados. Se tudo correr bem, ambos bccs válidos, então os dados são aceites para processamento (e enviados à Aplicação através do buffer passado como argumento) e a trama deve ser confirmada com o envio de uma trama de supervisão **RR (ACK positivo)** com a seguinte divisão:



Caso não seja detetado nenhum erro no cabeçalho da trama I, mas seja detetado um erro pelo BCC no campo de dados, este campo é descartado e é feito um pedido de retransmissão através de uma resposta **REJ (ACK negativo)**, o que nos permite antecipar a ocorrência de um time-out no emissor.

Nota 1: "BCC (Block Check Character) - permite a detecção de erros baseada na geração de um octeto (BCC) tal que exista um número par de 1s em cada posição (bit), considerando todos os octetos protegidos pelo BCC (cabeçalho ou dados, conforme os casos) e o próprio BCC (antes de stuffing)" - Guião do trabalho

3. llwrite()

Esta função recebe do **ApplicationLayer** a informação para ser enviada e o tamanho da mesma. Em primeiro lugar, fazemos byte stuffing da informação a ser enviada e criamos uma trama de informação ***I***. De seguida, a trama criada é enviada com o número de sequência correto, "0" ou "1". Após o envio, é inicializado um alarme e começa-se a leitura ou da trama RR ou da trama REJ, também com a ajuda de uma máquina de estados. No caso de receber a trama de supervisão REJ, é reenviada à informação. A função termina quando se recebe uma trama de supervisão RR ou quando o número de retransmissões do alarme for alcançado.

4. llclose()

Esta função recebe um booleano que verifica se é para mostrar as estatísticas.

O emissor envia um DISC, ativa o alarme e recebe outro DISC, que é lido com a ajuda de uma máquina de estados. No final envia um UA e caso tudo tenha corrido bem (nenhum alarme) a função termina.

O receptor recebe um DISC e lê-o com a ajuda de uma máquina de estados, envia um outro DISC e no final recebe um UA que também é lido recorrendo à máquina de estados.

Protocolo de aplicação

O protocolo de aplicação tem apenas a função *applicationLayer()* que recebe o serialPort que está a ser usado, a função da máquina (transmissor ou receptor), o

baudRate, o número de retransmissões permitidas, o timeout do alarme e ainda o nome do ficheiro que vai ser lido ou escrito.

Em primeiro lugar, tanto o receptor como o transmissor preenchem um elemento do tipo **struct LinkLayer** com os valores recebidos como parâmetros da função, e de seguida, é chamada a função *llopen()* com a **struct LinkLayer** devidamente preenchida.

Transmissor

O transmissor começa por abrir o ficheiro que vai ser enviado, e cria o pacote de controlo START que guarda a informação do nome e do tamanho do ficheiro. Depois vai lendo do ficheiro informação de tamanho igual ou inferior ao MAX_PAYLOAD_SIZE até já não haver mais informação para ler. Com esta informação é criado o pacote de dados que é enviado para a função *llwrite()*. No final é enviado o pacote de controlo END com a mesma informação do pacote de START. No final é chamada a função de *llclose()*.

Recetor

O receptor começa por abrir o ficheiro para escrita e por receber o pacote de controlo START através do *llread()*, guardando a informação recebida do nome do ficheiro e do seu tamanho. Em seguida, a função *llread()* é chamada continuamente, sendo que em cada iteração é verificado se o número de sequência está correto, e se estiver a informação é escrita no ficheiro. Quando é recebido o pacote de controlo END a leitura pára e verificamos se o nome e tamanho recebidos nesse pacote são iguais aos recebidos no pacote START. No final é chamada a função de *llclose()*.

Observação:
Tanto no protocolo de ligação de dados como no protocolo de aplicação são verificados valores de retorno e tratamento de erros.

Validação

Testes realizados:

- Envio de ficheiros de diferentes tamanhos
- Interrupção da ligação entre as portas séries ao ligar e desligar a mesma
- Geração de ruído na ligação entre as portas série
- Transmissão e receção de ficheiros com diferentes valores de baudrate
- Transmissão e receção de ficheiros com tramas de informação de diferente tamanho
- Transmissão e receção com a simulação de diferentes tempos de propagação

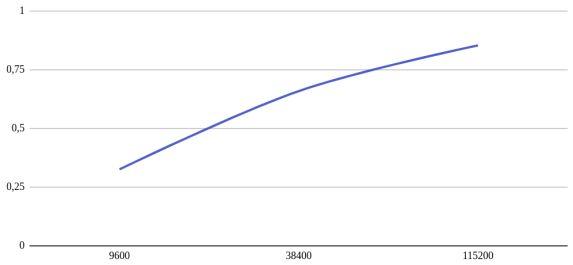
Todos os testes efetuados foram sucedidos e obtiveram o resultado previsto.

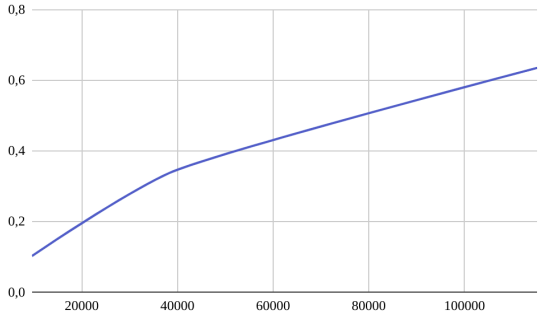
Eficiência do protocolo de ligação de dados

Nesta secção de análise de eficiência em diferentes casos, o gráfico da esquerda representa os valores teóricos e o gráfico da direita os valores práticos.

VARIAÇÃO DO BAUDRATE

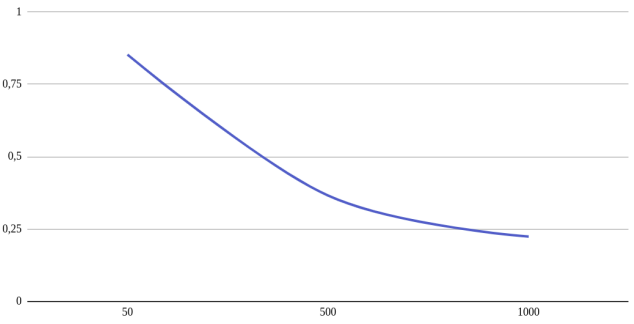
Através destes gráficos, concluímos que as duas curvas são bastante semelhantes, provando assim que com o aumento do baudrate, a eficiência também aumenta.

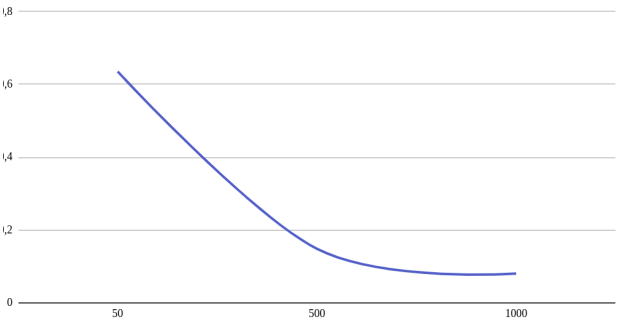




VARIAÇÃO DO TAMANHO DAS TRAMAS

Aqui mais uma vez podemos observar que as curvas são semelhantes, e o aumento do tamanho da trama leva a uma diminuição da eficiência na transmissão daquela trama, uma vez que está a enviar mais informação. No entanto, quando estamos a falar da eficiência do envio total das tramas, podemos concluir que esta vai aumentar porque vai haver menos tramas a ser enviadas.

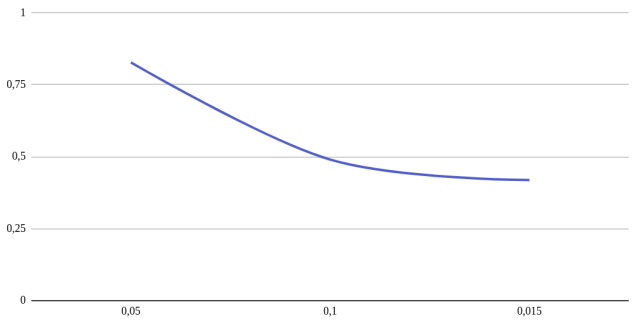


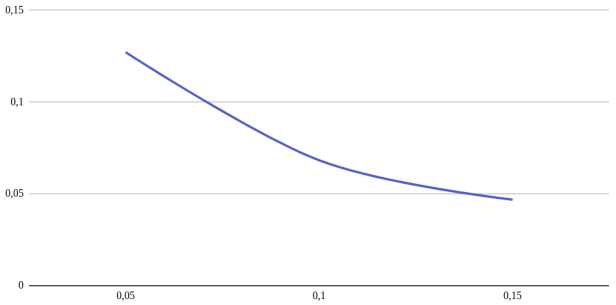


VARIAÇÃO DO TEMPO DE PROPAGAÇÃO

Novamente, os gráficos apresentam curvas semelhantes, pelo que podemos concluir que o aumento do tempo de propagação leva a uma diminuição da eficiência, uma vez que

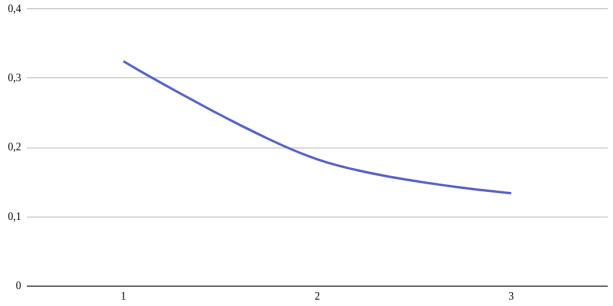
demoramos mais tempo a transferir um ficheiro. Utilizamos a função sleep() para aumentar o tempo de propagação do programa.

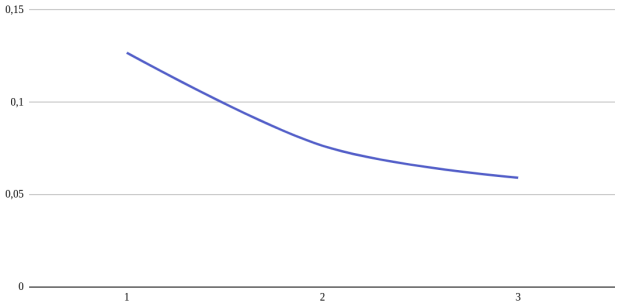




VARIAÇÃO DO ERRO

Por fim, aqui as curvas também são semelhantes, pelo que fica comprovado que com o aumento de erros (nº de alarmes), a nossa eficiência acaba por diminuir de forma logarítmica.





Conclusões

A realização deste trabalho tinha como objetivo, proporcionar-nos uma melhor compreensão do funcionamento de um protocolo de transferência de dados.

Assim, não só apreendemos como cada trama está organizada mas também a importância da interdependência entre camadas, uma vez que protocolo de ligação não recorre a qualquer processo do protocolo de aplicação.

Também nos apercebemos da complexidade existente por trás de uma operação tão simples e tão utilizada no dia a dia, a transferência de dados.

ANEXOS I - código fonte

link_layer.c

```c
// Link layer protocol implementation
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termios.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>
#include "link_layer.h"

#define FALSE 0
#define TRUE 1

#define BUF_SIZE 256
#define FLAG 0x7E
#define CSET 0x03
#define CUA 0x07
#define C0 0x00
#define C1 0x40
#define CDISC 0x0B
#define CRR0 0x05
#define CRR1 0x85
#define CREJ0 0x01
#define CREJ1 0x81
#define A 0x03
#define ESC 0x7D

////////////////////////////////////////////////
// LLOPEN
////////////////////////////////////////////////

typedef enum {
    START,
    FLAG_RCV,
    A_RCV,
    C_RCV,
    BCC_OK
} StateEnum;

clock_t start, end;
struct termios oldtio;
volatile int STOP = FALSE;
int alarmEnabled = FALSE;
int alarmCount = 0;
int fd;
int nTries = 0;
int timeout = 0;
int ns = 0;
int nr = 1;
LinkLayerRole role;

void alarmHandler(int signal) {
    alarmEnabled = FALSE;
    alarmCount++;
    printf("Alarm #%d\n", alarmCount);
    printf("enable: %d\n", alarmEnabled);
}


int llopen(LinkLayer connectionParameters) {
    role= connectionParameters.role;
    start = clock();

    fd = open(connectionParameters.serialPort, O_RDWR | O_NOCTTY);

    if (fd < 0) {
        perror(connectionParameters.serialPort);
```

```
        exit(-1);
    }
    nTries = connectionParameters.nRetransmissions;
    timeout = connectionParameters.timeout;

    struct termios newtio;

    // Save current port settings
    if (tcgetattr(fd, &oldtio) == -1) {
        perror("tcgetattr");
        exit(-1);
    }

    // Clear struct for new port settings
    memset(&newtio, 0, sizeof(newtio));

    newtio.c_cflag = connectionParameters.baudRate | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;


    if (connectionParameters.role == LlTx) {
        newtio.c_lflag = 0;
        newtio.c_cc[VTIME] = 0.1; // Inter-character timer unused
        newtio.c_cc[VMIN] = 0;  // Blocking read until 5 chars received[0]

        // VTIME e VMIN should be changed in order to protect with a
        // timeout the reception of the following character(s)

        // Now clean the line and activate the settings for the port
        // tcflush() discards data written to the object referred to
        // by fd but not transmitted, or data received[0] but not read,
        // depending on the value of queue_selector:
        //   TCIFLUSH - flushes data received[0] but not read.
        tcflush(fd, TCIOFLUSH);

        // Set new port settings
        if (tcsetattr(fd, TCSANOW, &newtio) == -1) {
            perror("tcsetattr");
            exit(-1);
        }
        unsigned char set[] = {FLAG, A, CSET, (A ^ CSET), FLAG};
        int done = FALSE;
        (void) signal(SIGALRM, alarmHandler);
        unsigned char a, c;
        StateEnum state = START;
        while (alarmCount < nTries && !done) {
            if (alarmEnabled == FALSE) {
                state = START;
                if (write(fd, set, 5) == -1) {
                    printf("error writing\n");
                    return -1;
                }
                alarm(timeout); // Set alarm to be triggered in 3s
                alarmEnabled = TRUE;
            }
            unsigned char received[1];
            int bytes = read(fd, received, 1);
            if (bytes == -1) {
                printf("error reading in llopen\n");
                return -1;
            }
            if (bytes == 0)
                continue;
            //printf("var=0x%02X\n", (unsigned int)(received[0] & 0xFF));

            switch (state) {
                case START:
                    if (received[0] == FLAG) {
                        state = FLAG_RCV;
                    }
                    break;
```

```
            case FLAG_RCV:
                if (received[0] == FLAG)
                    continue;
                else if (received[0] != A) {
                    state = START;
                } else {
                    a = received[0];
                    state = A_RCV;
                }
                break;
            case A_RCV:
                if (received[0] == FLAG) {
                    state = FLAG_RCV;
                } else if (received[0] != CUA) {
                    state = START;
                } else {
                    c = received[0];
                    state = C_RCV;
                }
                break;
            case C_RCV:
                if (received[0] != (a ^ c)) {
                    state = START;
                } else {
                    state = BCC_OK;
                }
                break;
            case BCC_OK:
                if (received[0] != FLAG) {
                    state = START;
                } else {
                    done = TRUE;
                }
                break;
        }

    }
    alarm(0);
    alarmEnabled = FALSE;
    alarmCount=0;

} else if (connectionParameters.role == LlRx) {
    newtio.c_lflag = 0;
    newtio.c_cc[VTIME] = 0; // Inter-character timer unused
    newtio.c_cc[VMIN] = 1;  // Blocking read until 5 chars received[0]

    // VTIME e VMIN should be changed in order to protect with a
    // timeout the reception of the following character(s)

    // Now clean the line and activate the settings for the port
    // tcflush() discards data written to the object referred to
    // by fd but not transmitted, or data received[0] but not read,
    // depending on the value of queue_selector:
    //   TCIFLUSH - flushes data received[0] but not read.
    tcflush(fd, TCIOFLUSH);

    // Set new port settings
    if (tcsetattr(fd, TCSANOW, &newtio) == -1) {
        perror("tcsetattr");
        exit(-1);
    }
    unsigned char set[5] = {0};
    StateEnum state = START;
    int running = 1;
    while (running) {
        unsigned char received[1];
        int bytes = read(fd, received, 1);
        if (bytes == -1) {
            printf("error reading in llopen2\n");
            return -1;
        }
        //printf("var=0x%02X\n", (unsigned int)(received[0] & 0xFF));
```

```
            switch (state) {
                case START:
                    if (received[0] == FLAG) {
                        set[0] = received[0];
                        state = FLAG_RCV;
                    }
                    break;
                case FLAG_RCV:
                    if (received[0] == FLAG)
                        continue;
                    else if (received[0] != A) {
                        state = START;
                    } else {
                        set[1] = received[0];
                        state = A_RCV;
                    }
                    break;
                case A_RCV:
                    if (received[0] == FLAG) {
                        state = FLAG_RCV;
                        set[0] = received[0];
                    } else if (received[0] != CSET) {
                        state = START;
                    } else {
                        set[2] = received[0];
                        state = C_RCV;
                    }
                    break;
                case C_RCV:
                    if (received[0] != (set[1] ^ set[2])) {
                        state = START;
                    } else {
                        set[3] = received[0];
                        state = BCC_OK;
                    }
                    break;
                case BCC_OK:
                    if (received[0] != FLAG) {
                        state = START;
                    } else {
                        set[4] = received[0];
                        running = 0;
                    }
                    break;
            }
        }
        unsigned char ua[] = {FLAG, A, CUA, (A ^ CUA), FLAG};
        if (write(fd, ua, 5) == -1) {
            printf("error writing\n");
            return -1;
        }
    }
    return 1;
}

////////////////////////////////////////////////
// LLWRITE
////////////////////////////////////////////////
int llwrite(const unsigned char *buf, int bufSize) {
    int done = FALSE;
    (void) signal(SIGALRM, alarmHandler);
    StateEnum state = START;

    unsigned int count = 0;


    for (int k = 0; k < bufSize; k++) {
        if (buf[k] == FLAG || buf[k] == ESC) {
            count += 1;
        }
    }
```

```
    unsigned int size = bufSize + 6 + count + 1;
    unsigned char i[size];
    for (int k = 0; k < size; k++) {
        i[k] = 0;
    }
    // write info;
    i[0] = FLAG;
    i[1] = A;
    if (ns == 0) {
        i[2] = C0;
        i[3] = A ^ C0;
    } else if (ns == 1) {
        i[2] = C1;
        i[3] = A ^ C1;
    }
    unsigned int bcc2 = 0, k = 0;
    for (int j = 0; j < bufSize; j++) {
        if (buf[j] == FLAG) {
            i[j + 4 + k] = ESC;
            i[j + 5 + k] = 0x5E;
            bcc2 ^= ESC;
            bcc2 ^= 0x5E;
            k++;

        } else if (buf[j] == ESC) {
            i[j + 4 + k] = ESC;
            i[j + 5 + k] = 0x5D;
            bcc2 ^= ESC;
            bcc2 ^= 0x5D;
            k++;

        } else {
            i[j + 4 + k] = buf[j];
            bcc2 ^= buf[j];
        }

    }
    if(bcc2==FLAG){
        i[size - 1] = FLAG;
        i[size - 2] = 0x5E;
        i[size - 3] = ESC;
    } else if(bcc2 == ESC){
        i[size - 1] = FLAG;
        i[size - 2] = 0x5D;
        i[size - 3] = ESC;
    }else{
        size--;
        i[size - 1]= FLAG;
        i[size - 2] = bcc2;
    }

    /*
    for (int l = 0; l < size; l++) {
        printf("aa=0x%02X, unsigned char: %c\n", (unsigned int)(i[l] & 0xFF), i[l]);
    }
    printf("size: %d\n", count);
    printf("size: %d\n", size);*/
    while (alarmCount < nTries && !done) {


        if (alarmEnabled == FALSE) {
            state = START;
            if (write(fd, i, size) == -1) {
                printf("error writing\n");
                return -1;
            }
            alarm(timeout); // Set alarm to be triggered in 3s
            alarmEnabled = TRUE;
        }
        unsigned char received[1];
        int bytes = read(fd, received, 1);
        if (bytes == -1) {
```

```
            printf("error reading in llwrite\n");
            return -1;
        }
        //printf("var=0x%02X\n", (unsigned int)(received[0] & 0xFF));
        unsigned char a, c;
        switch (state) {
            case START:
                if (received[0] == FLAG) {
                    state = FLAG_RCV;
                }
                break;
            case FLAG_RCV:
                if (received[0] == FLAG)
                    continue;
                else if (received[0] != A) {
                    state = START;
                } else {
                    a = received[0];
                    state = A_RCV;
                }
                break;
            case A_RCV:
                if (received[0] == FLAG) {
                    state = FLAG_RCV;
                } else if (received[0] == CRR0){
                    if(ns==0){
                        if (write(fd, i, size) == -1) {
                            printf("error writing\n");
                            return -1;
                        }
                        state=START;
                    }else {
                        c = received[0];
                        state = C_RCV;
                    }
                } else if (received[0] == CRR1) {
                    if(ns==1){
                        if (write(fd, i, size) == -1) {
                            printf("error writing\n");
                            return -1;
                        }
                        state=START;
                    }else {
                        c = received[0];
                        state = C_RCV;
                    }
                } else if (received[0] == CREJ0 || received[0] == CREJ1) {
                    if (write(fd, i, size) == -1) {
                        printf("error writing\n");
                        return -1;
                    }
                    state = START;
                } else {
                    state = START;
                }
                break;
            case C_RCV:
                if (received[0] != (a ^ c)) {
                    state = START;
                } else {
                    state = BCC_OK;
                }
                break;
            case BCC_OK:
                if (received[0] != FLAG) {
                    state = START;
                } else {
                    alarm(0);
                    alarmEnabled = FALSE;
                    done = TRUE;
                }
                break;
```

```
        }

    }
    if(!done){
        printf("alarm count reached\n");
        return -1;
    }

    if (ns == 0)
        ns = 1;
    else if (ns == 1)
        ns = 0;
    alarm(0);
    alarmEnabled = FALSE;
    alarmCount=0;
    return size;
}

////////////////////////////////////////////////
// LLREAD
////////////////////////////////////////////////
int llread(unsigned char *packet) {
    StateEnum state = START;
    int running = TRUE;
    int esc_activated = FALSE;
    int data_size = 0;
    unsigned char bcc2 = 0;
    while (running) {
        //Verifies if everything is okay with byte received
        unsigned char received[1];
        int byte = read(fd, received, 1);
        if (byte == -1) {
            printf("error reading in llread\n");
            return -1;
        }
        unsigned char a, c;
        switch (state) {
            case START:
                if (received[0] == FLAG) {
                    //printf("flag=0x%02Xunsigned char: %c\n", (unsigned int)(received[0] & 0xFF), received[0]);
// print do que recebe
                    state = FLAG_RCV;
                }
                break;
            case FLAG_RCV:
                if (received[0] == FLAG)
                    continue;
                else if (received[0] != A) {
                    state = START;
                } else {
                    //printf("a=0x%02Xunsigned char: %c\n", (unsigned int)(received[0] & 0xFF), received[0]); //
print do que recebe
                    a = received[0];
                    state = A_RCV;
                }
                break;
            case A_RCV:
                if (received[0] == FLAG) {
                    state = FLAG_RCV;
                } else if (received[0] == C0){
                    if(nr==0){
                        unsigned char ack[5] = {0};
                        ack[0] = FLAG;
                        ack[1] = A;
                        ack[4] = FLAG;
                        if (nr == 0) {
                            ack[2] = CRR0;
                            ack[3] = A ^ CRR0;
                        } else if (nr == 1) {
                            ack[2] = CRR1;
                            ack[3] = A ^ CRR1;
                        }
```

```
                        if (write(fd, ack, 5) == -1) {
                            printf("error writing\n");
                            return -1;
                        }
                        state=START;
                    }else {
                        c = received[0];
                        state = C_RCV;
                    }

                } else if (received[0] == C1) {
                    if(nr==1){
                        unsigned char ack[5] = {0};
                        ack[0] = FLAG;
                        ack[1] = A;
                        ack[4] = FLAG;
                        if (nr == 0) {
                            ack[2] = CRR0;
                            ack[3] = A ^ CRR0;
                        } else if (nr == 1) {
                            ack[2] = CRR1;
                            ack[3] = A ^ CRR1;
                        }
                        if (write(fd, ack, 5) == -1) {
                            printf("error writing\n");
                            return -1;
                        }
                        state=START;
                    }else {
                        c = received[0];
                        state = C_RCV;
                    }
                } else {
                    state = START;
                }
                break;
            case C_RCV:
                if (received[0] == FLAG) {
                    state = FLAG_RCV;
                } else if (received[0] != (a ^ c)) {
                    state = START;
                } else {
                    //printf("bcc=0x%02Xunsigned char: %c\n", (unsigned int)(received[0] & 0xFF), received[0]); //
print do que recebe
                    state = BCC_OK;
                }
                break;
            case BCC_OK:
                if (received[0] == FLAG) {
                    //printf("\nbcc1=0x%02X, unsigned char: %c\n", (unsigned int)(packet[data_size-1] & 0xFF),
packet[data_size]);
                    if(packet[data_size - 1] == ESC){
                        bcc2=bcc2^ESC^0x5D;
                    }
                    else if(packet[data_size - 1] == FLAG){
                        bcc2=bcc2^ESC^0x5E;
                    }
                    else{
                        bcc2 ^= packet[data_size - 1];
                    }
                    if (packet[data_size - 1] == (bcc2)) {
                        packet[data_size - 1] = 0;
                        data_size--;
                        running = FALSE;
                        unsigned char ack[5] = {0};
                        ack[0] = FLAG;
                        ack[1] = A;
                        ack[4] = FLAG;
                        if (nr == 0) {
                            ack[2] = CRR0;
                            ack[3] = A ^ CRR0;
                        } else if (nr == 1) {
```

```
                        ack[2] = CRR1;
                        ack[3] = A ^ CRR1;
                    }
                    if (write(fd, ack, 5) == -1) {
                        printf("error writing\n");
                        return -1;
                    }

                } else {
                    state = START;
                    data_size = 0;
                    bcc2 = 0;
                    unsigned char nack[5] = {0};
                    nack[0] = FLAG;
                    nack[1] = A;
                    nack[4] = FLAG;
                    if (nr == 0) {
                        nack[2] = CREJ0;
                        nack[3] = A ^ CREJ0;
                    } else if (nr == 1) {
                        nack[2] = CREJ1;
                        nack[3] = A ^ CREJ1;
                    }
                    if (write(fd, nack, 5) == -1) {
                        printf("error writing\n");
                        return -1;
                    }
                }
            } else {
                if (data_size >= MAX_PAYLOAD_SIZE + 1) {
                    printf("fuck=0x%02Xunsigned char: %c\n", (unsigned int)(received[0] & 0xFF), received[0]);
// print do que recebe
                    unsigned char nack[5] = {0};
                    nack[0] = FLAG;
                    nack[1] = A;
                    nack[4] = FLAG;
                    if (nr == 0) {
                        nack[2] = CREJ0;
                        nack[3] = A ^ CREJ0;
                    } else if (nr == 1) {
                        nack[2] = CREJ1;
                        nack[3] = A ^ CREJ1;
                    }
                    data_size = 0;
                    if (write(fd, nack, 5) == -1) {
                        printf("error writing\n");
                        return -1;
                    }
                    state = START;
                    bcc2 = 0;
                    continue;
                }
                if (received[0] == ESC) {
                    esc_activated = TRUE;
                    continue;
                }
                if (esc_activated) {
                    if (received[0] == 0x5E) {
                        packet[data_size] = FLAG;
                        data_size++;
                        bcc2 ^= ESC;
                        bcc2 ^= 0x5E;
                    }
                    if (received[0] == 0x5D) {
                        packet[data_size] = ESC;
                        data_size++;
                        bcc2 ^= ESC;
                        bcc2 ^= 0x5D;
                    }
                    esc_activated = FALSE;
                } else {
                    //printf("get=0x%02Xunsigned char: %c\n", (unsigned int)(received[0] & 0xFF),
```

```
received[0]); // print do que recebe
                        packet[data_size] = received[0];
                        data_size++;
                        bcc2 ^= received[0];
                    }

                }

                break;

        }


    }
    if (nr == 0)
        nr = 1;
    else if (nr == 1)
        nr = 0;

    return data_size;
}

////////////////////////////////////////////////
// LLCLOSE
////////////////////////////////////////////////
int llclose(int showStatistics) {
    if (role == LlTx) {

        unsigned char disc[] = {FLAG, A, CDISC, (A ^ CDISC), FLAG};
        int done = FALSE;
        (void) signal(SIGALRM, alarmHandler);
        unsigned char a, c;
        StateEnum state = START;
        while (alarmCount < nTries && !done) {
            if (alarmEnabled == FALSE) {
                state = START;
                if (write(fd, disc, 5) == -1) {
                    printf("error writing\n");
                    return -1;
                }
                alarm(timeout); // Set alarm to be triggered in 3s
                alarmEnabled = TRUE;
            }
            unsigned char received[1];
            int bytes = read(fd, received, 1);
            if (bytes == -1) {
                printf("error reading in llopen\n");
                return -1;
            }
            if (bytes == 0)
                continue;
            printf("var=0x%02X\n", (unsigned int)(received[0] & 0xFF));

            switch (state) {
                case START:
                    if (received[0] == FLAG) {
                        state = FLAG_RCV;
                    }
                    break;
                case FLAG_RCV:
                    if (received[0] == FLAG)
                        continue;
                    else if (received[0] != A) {
                        state = START;
                    } else {
                        a = received[0];
                        state = A_RCV;
                    }
                    break;
                case A_RCV:
                    if (received[0] == FLAG) {
                        state = FLAG_RCV;
```

```
                } else if (received[0] != CDISC) {
                    state = START;
                } else {
                    c = received[0];
                    state = C_RCV;
                }
                break;
            case C_RCV:
                if (received[0] != (a ^ c)) {
                    state = START;
                } else {
                    state = BCC_OK;
                }
                break;
            case BCC_OK:
                if (received[0] != FLAG) {
                    state = START;
                } else {
                    done = TRUE;
                }
                break;
        }

    }
    alarm(0);
    alarmEnabled = FALSE;
    unsigned char ua[] = {FLAG, A, CUA, (A ^ CUA), FLAG};
    if (write(fd, ua, 5) == -1) {
        printf("error writing\n");
        return -1;
    }

} else if (role == LlRx) {

    unsigned char set[5] = {0};
    StateEnum state = START;
    int running = 1;
    while (running) {
        unsigned char received[1];
        int bytes = read(fd, received, 1);
        if (bytes == -1) {
            printf("error reading in llopen2\n");
            return -1;
        }
        printf("var=0x%02X\n", (unsigned int)(received[0] & 0xFF));
        switch (state) {
            case START:
                if (received[0] == FLAG) {
                    set[0] = received[0];
                    state = FLAG_RCV;
                }
                break;
            case FLAG_RCV:
                if (received[0] == FLAG)
                    continue;
                else if (received[0] != A) {
                    state = START;
                } else {
                    set[1] = received[0];
                    state = A_RCV;
                }
                break;
            case A_RCV:
                if (received[0] == FLAG) {
                    state = FLAG_RCV;
                    set[0] = received[0];
                } else if (received[0] != CDISC) {
                    state = START;
                } else {
                    set[2] = received[0];
                    state = C_RCV;
                }
```

```
                break;
            case C_RCV:
                if (received[0] != (set[1] ^ set[2])) {
                    state = START;
                } else {
                    set[3] = received[0];
                    state = BCC_OK;
                }
                break;
            case BCC_OK:
                if (received[0] != FLAG) {
                    state = START;
                } else {
                    set[4] = received[0];
                    running = 0;
                }
                break;
        }
    }
    unsigned char disc[] = {FLAG, A, CDISC, (A ^ CDISC), FLAG};
    if (write(fd, disc, 5) == -1) {
        printf("error writing\n");
        return -1;
    }
    while (running) {
        unsigned char received[1];
        int bytes = read(fd, received, 1);
        if (bytes == -1) {
            printf("error reading in llopen2\n");
            return -1;
        }
        printf("var=0x%02X\n", (unsigned int)(received[0] & 0xFF));
        switch (state) {
            case START:
                if (received[0] == FLAG) {
                    set[0] = received[0];
                    state = FLAG_RCV;
                }
                break;
            case FLAG_RCV:
                if (received[0] == FLAG)
                    continue;
                else if (received[0] != A) {
                    state = START;
                } else {
                    set[1] = received[0];
                    state = A_RCV;
                }
                break;
            case A_RCV:
                if (received[0] == FLAG) {
                    state = FLAG_RCV;
                    set[0] = received[0];
                } else if (received[0] != CUA) {
                    state = START;
                } else {
                    set[2] = received[0];
                    state = C_RCV;
                }
                break;
            case C_RCV:
                if (received[0] != (set[1] ^ set[2])) {
                    state = START;
                } else {
                    set[3] = received[0];
                    state = BCC_OK;
                }
                break;
            case BCC_OK:
                if (received[0] != FLAG) {
                    state = START;
                } else {
```

```
                        set[4] = received[0];
                        running = 0;
                    }
                    break;
                }
            }

        }


    end = clock();
    double time_elapsed = ((double) (end - start)) / CLOCKS_PER_SEC;
    if (tcsetattr(fd, TCSANOW, &oldtio) == -1) {
        perror("tcsetattr");
        exit(-1);
    }

    if (showStatistics == 1) {
        printf("Time Spent: %f seconds\n", time_elapsed);
    }

    close(fd);


    return 1;
}
```

application_layer.c

```
// Application layer protocol implementation

#include "application_layer.h"
#include "link_layer.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <limits.h>


int numDigits(long n) {
    int count = 0;
    do {
        n /= 10;
        ++count;
    } while (n != 0);
    return count;
}

void applicationLayer(const char *serialPort, const char *role, int baudRate,
                      int nTries, int timeout, const char *filename) {
    LinkLayer llayer;
    llayer.serialPort = serialPort;
    if (!strcmp(role, "tx"))
        llayer.role = LlTx;
    else if (!strcmp(role, "rx"))
        llayer.role = LlRx;
    llayer.timeout = timeout;
    llayer.baudRate = baudRate;
```

```
    llayer.nRetransmissions = nTries;
    llopen(llayer);

    if (llayer.role == LlTx) {

        printf("Sending data...\n");

        // calculating the size of the file
        struct stat st;
        stat(filename, &st);
        off_t fileSize = st.st_size;


        int fileSize_bytes = numDigits(fileSize);
        char fileSizeStr[fileSize_bytes];
        sprintf(fileSizeStr, "%ld", fileSize);

        int fileName_bytes = strlen(filename);

        unsigned int controlPacketSize = 3 + fileSize_bytes + 2 + fileName_bytes;
        unsigned char controlPacket[controlPacketSize];
        controlPacket[0] = 2;
        controlPacket[1] = 0;
        controlPacket[2] = fileSize_bytes;
        for (int i = 0; i < fileSize_bytes; i++) {
            controlPacket[i + 3] = fileSizeStr[i];
        }
        controlPacket[3 + fileSize_bytes] = 1;
        controlPacket[3 + fileSize_bytes + 1] = fileName_bytes;
        for (int i = 0; i < fileName_bytes; i++) {
            controlPacket[3 + fileSize_bytes + 2 + i] = filename[i];
        }

        if (llwrite(controlPacket, controlPacketSize) == -1) {
            printf("error in llwrite\n");
            return;
        }

        FILE *file;
        file = fopen(filename, "rb");
        if (NULL == file) {
            printf("file can't be opened \n");
        }
        int n = 0;
        unsigned char dataPacket[MAX_PAYLOAD_SIZE] = {0};
        dataPacket[0] = 1;
        while (TRUE) {
            if (fileSize >= MAX_PAYLOAD_SIZE - 4) {
                unsigned char buffer[MAX_PAYLOAD_SIZE - 4] = {0};
                fread(buffer, MAX_PAYLOAD_SIZE - 4, 1, file);
                fileSize = fileSize - (MAX_PAYLOAD_SIZE - 4);
                dataPacket[1] = n;
                dataPacket[3] = (MAX_PAYLOAD_SIZE - 4) / 256;
                dataPacket[2] = (MAX_PAYLOAD_SIZE - 4) % 256;
                for (int i = 0; i < MAX_PAYLOAD_SIZE - 4; i++) {
                    dataPacket[i + 4] = buffer[i];
                }
                n++;
                if (llwrite(dataPacket, MAX_PAYLOAD_SIZE) == -1) {
                    printf("error in llwrite\n");
                    return;
                }
            } else if (fileSize == 0) {
```

```
                break;
            } else {
                unsigned char buffer[fileSize];
                if (!fread(buffer, fileSize, 1, file)) {
                    printf("error reading file\n");
                    break;
                }

                dataPacket[1] = n;
                dataPacket[3] = (fileSize) / 256;
                dataPacket[2] = (fileSize) % 256;
                for (int i = 0; i < fileSize; i++) {
                    dataPacket[i + 4] = buffer[i];
                }
                n++;
                if (llwrite(dataPacket, (int) fileSize + 4) == -1) {
                    printf("error in llwrite\n");
                    return;
                }
                break;
            }
        }

        // Closing the file
        fclose(file);

        controlPacket[0] = 3;
        if (llwrite(controlPacket, controlPacketSize) == -1) {
            printf("error in llwrite\n");
            return;
        }

        printf("File sent!\n");

    } else if (llayer.role == LlRx) {
        FILE *file;
        file = fopen(filename, "wb");
        unsigned char buf[MAX_PAYLOAD_SIZE] = {0};
        unsigned char name[MAX_PAYLOAD_SIZE] = {0};
        unsigned char nameSize;
        unsigned char fileLength;
        unsigned char fileSize[MAX_PAYLOAD_SIZE];
        int count = 0;
        printf("Receiving data...\n");
        int size = llread(buf);
        if (size == -1) {
            printf("Error reading start control packet.\n");
        }
        if (buf[0] == 2) {
            if (buf[1] == 0) {
                fileLength = buf[2];
                for (int i = 0; i < fileLength; i++) {
                    fileSize[i] = buf[i + 3];

                }
            }
            if (buf[3 + fileLength] == 1) {
                nameSize = buf[4 + fileLength];
                for (int i = 0; i < nameSize; i++) {
                    name[i] = buf[i + 5 + fileLength];
                }
            }
            while (TRUE) {
```

```
            unsigned char info[MAX_PAYLOAD_SIZE] = {0};
            size = llread(info);

            if (size == -1) {
                printf("Error reading control/data packet.\n");
            }
            if (info[0] == 3) {
                unsigned char name2[MAX_PAYLOAD_SIZE] = {0};
                unsigned char nameSize2;
                unsigned char fileLength2;
                unsigned char fileSize2[MAX_PAYLOAD_SIZE];
                if (info[1] == 0) {
                    fileLength2 = info[2];
                    for (int i = 0; i < fileLength2; i++) {
                        fileSize2[i] = info[i + 3];

                    }
                }
                if (fileLength != fileLength2) {
                    printf("Start info and ending info don't match.\n");
                    return;
                }
                for (int i = 0; i < fileLength; i++) {
                    if (fileSize[i] != fileSize2[i]) {
                        printf("Start info and ending info don't match.\n");
                        return;
                    }
                }
                if (info[3 + fileLength2] == 1) {
                    nameSize2 = info[4 + fileLength2];
                    for (int i = 0; i < nameSize2; i++) {
                        name2[i] = info[i + 5 + fileLength2];
                    }
                }
                if (nameSize != nameSize2) {
                    printf("Start info and ending info don't match.\n");
                    return;
                }
                for (int i = 0; i < nameSize; i++) {
                    if (name[i] != name2[i]) {
                        printf("Start info and ending info don't match.\n");
                        return;
                    }
                }
                break;
            }
            unsigned char n = info[1];
            if (count != n) {
                printf("sequence of files received not correct, please try again\n");
            }
            count++;
            unsigned char l1 = info[2];
            unsigned char l2 = info[3];
            unsigned long long k = l2 * 256 + l1;
            unsigned char data[k];
            for (int i = 0; i < k; i++) {
                data[i] = info[i + 4];
            }
            fwrite(data, k, 1, file);
        }
    }

    printf("File received! \n");
```

```
    }


    if (llclose(TRUE) == -1) {
        printf("error in llclose\n");
    }

}
```