

**Rede de Computadores**

***1º Trabalho Laboratorial***

Mestrado Integrado em Engenharia Informática e Computação

André Esteves [up201606673@fe.up.pt](mailto:up201606673@fe.up.pt)

Luís Diogo Silva [up201503730@fe.up.pt](mailto:up201503730@fe.up.pt)

Francisco Friande [up201508213@fe.up.pt](mailto:up201508213@fe.up.pt)

**Índice**

[**Sumário** 3](#_Toc528951774)

[**Introdução** 3](#_Toc528951775)

[**Arquitetura** 4](#_Toc528951776)

[**Estrutura do código** 4](#_Toc528951777)

[**llwrite.c** 4](#_Toc528951778)

[**llread.c** 4](#_Toc528951779)

[**llopen.c** 4](#_Toc528951780)

[**llclose.c** 5](#_Toc528951781)

[**protocol.c** 5](#_Toc528951782)

[**receiver.c** 5](#_Toc528951783)

[**sender.c** 6](#_Toc528951784)

[**Casos de uso principais** 6](#_Toc528951785)

[**Protocolo de ligação lógica** 6](#_Toc528951786)

[**Protocolo de aplicação** 6](#_Toc528951787)

[**Validação** 6](#_Toc528951788)

[**Eficiência do protocolo de ligação de dados** 7](#_Toc528951789)

[**Conclusões** 7](#_Toc528951790)

[**Anexo I** 7](#_Toc528951791)

[**Makefile** 7](#_Toc528951792)

[**llclose.h** 7](#_Toc528951793)

[**llclose.c** 7](#_Toc528951794)

[**llopen.h** 9](#_Toc528951795)

[**llopen.c** 9](#_Toc528951796)

[**llread.h** 11](#_Toc528951797)

[**llread.c** 11](#_Toc528951798)

[**llwrite.h** 13](#_Toc528951799)

[**llwrite.c** 13](#_Toc528951800)

[**protocol.h** 15](#_Toc528951801)

[**protocol.c** 16](#_Toc528951802)

[**receiver.c** 24](#_Toc528951803)

[**sender.c** 28](#_Toc528951804)

[**Anexo II** 33](#_Toc528951805)

# **Sumário**

Este relatório foi elaborado no âmbito da unidade curricular de Rede de Computadores consistindo no desenvolvimento de uma aplicação capaz de transferir ficheiros de um computador para outro através de uma porta de série (RS-232) seguindo um protocolo de ligação de dados.

O trabalho foi realizado no seu todo no ambiente disponibilizado, sendo concluído com sucesso em todos os aspetos, cumprindo os objetivos pedidos.

# **Introdução**

O objetivo deste relatório é expor os aspetos mais teóricos da realização do projeto. O objetivo do trabalho era implementar um dado protocolo de ligação de dados, especificado no guião, de forma a fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por uma porta de série.

Para isto, foi necessário desenvolver funções de criação e sincronização de tramas, estabelecimento e terminação da ligação, confirmação de receção de uma trama sem erros, controlo de erros e de fluxo.

O relatório está composto pela seguinte forma:

* Arquitetura – Demonstração dos blocos funcionais e interfaces.
* Estrutura do código – Exposição das APIs, principais estruturas de dados, principais funções e sua relação com a arquitetura)
* Casos de uso principais – Identificação dos casos de uso principais e sequências de chamada de funções.
* Protocolo de ligação lógica – Identificação dos principais aspetos funcionais e descrição da estratégia de implementação destes aspetos.
* Protocolo de aplicação – Identificação dos principais aspetos funcionais; descrição da estratégia de implementação destes aspetos.
* Validação – Descrição dos testes efetuados com apresentação quantificada dos resultados.
* Eficiência do protocolo de ligação de dados – Caraterização estatística da eficiência do protocolo, feita com recurso a medidas sobre o código desenvolvido.
* Elementos de Valorização – Identificação dos elementos de valorização implementados e descrição da estratégia de implementação.
* Conclusão – Síntese da informação apresentada anteriormente e reflexão sobre os objetivos de aprendizagem alcançados.

# **Arquitetura**

# **Estrutura do código**

O código está dividido em vários ficheiros, llwrite.c, llread.c, llopen.c, llclose.c e protocol.c, sendo a base da aplicação, existindo um header file para cada um destes ficheiros onde estão declaradas todas as funções necessárias. Existe também sender.c, onde estão definidas as funções do emissor e o ficheiro receiver.c onde estão definidas as funções do recetor.

## **llwrite.c**

* **llwrite(…)**  
  - Calcula o BCC2 do package a enviar.

- Faz o stuff do package para ser enviado e devidamente interpretado.

- Envia a mensagem e espera pela confirmação num alarme de três segundos, com um limite de 3 tentativas de erro.

- Retornando “2” se ocorreu erro (envio de 3 mensagens consecutivas sem sucesso)

- Retornando sucesso (0 ou 1) de acordo com o valor do argumento flag. Se flag == 1 -> retorna 0, e se flag == 0 -> retorna 1. Verificando assim o correto envio da trama.

## **llread.c**

* **checkBCC2(…)**- Valida o conteúdo da trama.

- Retorna 0 em sucesso, 1 insucesso.

* **destuffing(…)**  
  - Descodifica a trama enviada retornando o package enviado pelo emissor.
* **Llread(…)**

- Lê uma trama enviada pelo emissor, verificando todos os erros na transmissão, seguindo do destuffing da mesma.

- Envia uma mensagem de retorno ao emissor a confirmar o correto envio ou a falha que ouve.

- Retorna o tamanho da mensagem recebida.

## **llopen.c**

* **llopen(…)**  
  - De acordo com a flag recebida é chamada a função correta, se for RECEIVER é usada a função llopen\_Receiver e o mesmo para a flag SENDER que é usada a função llopen\_Sender.
* **llopen\_Receiver(…)**  
  - Recebe uma trama verifica-a e envia a mensagem de sucesso ao emissor e retornando 0.

- Em caso de erros na trama é retornado -5.

* **llopen\_Sender(…)**  
  - Envia a trama ao recetor recebendo a mensagem de validação, caso não receba, tenta de novo em três possíveis tentativas.

- Caso falhe três vezes é retornado erro (2).

- Caso seja sucesso é verificada a mensagem de validação recebida pelo recetor, retornando 0 em sucesso, -6 em insucesso.

## **llclose.c**

* **llclose(…)**  
  - De acordo com a flag recebida é executado uma das duas funções seguintes. Se flag == RECEIVER llclose\_Receiver(…) é executada, senão llclose\_Sender(…).
* **llclose\_Receiver(…)**  
   -
* **llclose\_Sender(…)**  
   -

## **protocol.c**

* **attend(…)**
* **disableAlarm(…)**
* **read\_message(…)**
* **write\_message(…)**
* **parseMessageType(…)**
* **calculateBCC2(…)**
* **stuffing\_data\_package(…)**
* **stuffing\_control\_package(…)**
* **stuffing(…)**
* **heading(…)**

## **receiver.c**

* **setup(…)**
* **parseMessageStart(…)**
* **parseMessageData(…)**
* **saveData(…)**
* **createFile(…)**
* **main(…)**

## **sender.c**

* **setup(…)**
* **readFile(…)**
* **controlPackage(…)**
* **dataPackage(…)**
* **main(…)**

# **Casos de uso principais**

Os principais casos de uso desta aplicação são:

* Interface, que permite ao transmissor escolher o ficheiro a enviar, transferindo esse mesmo via porta série (/dev/ttyS0).
* Transmissor.
* Recetor.

Por forma a dar início à aplicação, sendo emissor, deverá inserir qual o ficheiro a ser enviado (ex: pinguim.gif). Sendo recetor basta iniciar o programa (iniciado com a porta /dev/ttyS0) ao qual vai tentar estabelecer ligação com o transmissor.

A transmissão dos dados é feita da seguinte forma:

* Transmissor escolhe ficheiro a enviar.
* Configuração da ligação entre os dois computadores.
* Estabelecimento da ligação (llopen).
* Transmissor envia dados.
* Recetor recebe dados e envia confirmação.
* Em cada de erro no envio, é enviado de novo, até a uma terceira tentativa falhada, terminando o processo com insucesso.
* Recetor guarda os dados num ficheiro com o mesmo nome e tipo que o enviado pelo emissor.
* Terminação da ligação (llclose).

# **Protocolo de ligação lógica**

* **llopen**

O propósito desta função é estabelecer a ligação entre o *sender* e o *receiver*. A função é igual para os dois, apresar de ter comportamento diferente mediante a *flag* que lhe for passada por argumento, isto é, dependendo se foi chamada pelo *sender* ou pelo *receiver*.

Quando chamada pelo *sender*, llopen começa por enviar a trama SET para tentar estabelecer a ligação com o *receiver*. Aguarda depois pela resposta do *receiver*, na forma de uma trama UA, procedendo ao reenvio periódico da trama SET até um número máximo de vezes, após as quais o programa termina.

Quando chamada pelo *receiver*, llopen começa por esperar a trama SET por parte do *sender*. Depois de a receber e de a analisar, certificando-se de que era a trama SET que esperava, envia a trama UA ao *sender*.

Para realizar a escrita, é a utilizada a função **write\_message**, que permite enviar uma mensagem contida num *buffer* para um determinado descritor de ficheiro. Para realizar a leitura, é utilizada a função **read\_message**, que recebe um caracter de cada vez de um determinado descritor de ficheiro. Esta função faz uso de uma máquina de estados para se certificar que recebe a mensagem na sua integridade. Finalmente, para analisar a mensagem recebida é utilizada a função **parseMessageType**, que verifica que a mensagem recebida não contém erros de transmissão e retorna o tipo de mensagem recebida.

* **llwrite**

Esta função é responsável pelo envio de pacotes de dados e de controlo do *sender*para o *receiver*. Cada invocação de llwrite envia apenas um pacote.

Esta função começa por preparar o pacote que lhe foi passado como argumento, calculando o *byte* BCC2, efetuando o *stuffing*, ou seja, a substituição de carateres protegidos que não podem ser enviados diretamente, e colocando o cabeçalho e a *flag* de terminação na mensagem. Depois disto, é enviada o pacote para o recetor. A função aguarda depois pela resposta do *receiver*, na forma de uma trama RR, procedendo ao reenvio periódico do pacote até um número máximo de vezes, após as quais o programa termina. O reenvio da mensagem pode ser efetuado antes do previsto caso seja recebida uma mensagem por parte do *receiver* que indique que houve erros de transmissão, tal como uma trama REJ ou uma trama RR com a *flag* oposta à esperada.

Para calcular o *byte* BCC2, é utilizada a função **calculateBBC2**, que o calcula mediante o pacote que lhe for passado por argumento. Para proceder ao *stuffing*, é utilizada a função **stuffing**, que recebe o pacote e o BBC2 calculado e retorna a mensagem já com o BCC2 concatenado e já com o *stuffing* realizado. Para realizar a colocação do cabeçalho e da *flag* de terminação, é utilizada a função **heading**, que retorna a mensagem já pronta a enviar. São também usadas as funções **write\_message**, **read\_message** e **parseMessageType** com usos semelhantes aos necessários durante a sua utilização na função llopen.

* **llread**

Esta função é responsável pela receção por parte do *receiver* de pacotes de dados e de controlo enviados pelo *sender***.** Cada invocação de llread recebe apenas um pacote.

Esta função começa por esperar a receção de um pacote por parte do *sender*. Depois realiza a verificação de erros na receção da mensagem, enviando a trama REJ caso encontre algum. Caso isso não aconteça, é realizado o *destuffing* da mensagem, isto é, o processo inverso ao realizado durante o llwrite, e a mensagem recebida é guardada para poder ser posteriormente escrita em memória. Após a mensagem ser aceite é enviado a trama RR ao *sender*.

Para realizar o *destuffing*, é utilizada a função **destuffing**, que retorna a mensagem completa como estava antes de o *stuffing* ter sido realizado. Para calcular a integridade da mensagem recebida, particularmente a validade do BCC2, é utilizada a função **checkBBC2**. São também utilizadas com fins semelhantes aos já descritos anteriormente as funções **write\_message** e **read\_message**.

* **llclose**

O propósito desta função é terminar a ligação entre o *sender* e o *receiver*. De forma análoga à função llopen, esta é igual para os dois, apresar de ter comportamento diferente mediante a flag que lhe for passada por argumento, isto é, dependendo se foi chamada pelo *sender* ou pelo *receiver*.

Quando chamada pelo *sender*, esta função começa por enviar uma trama DISC, espera a resposta por parte do *receiver*, na forma de uma trama DISC, e, após a receção desta, envia uma trama UA. O comportamento da função quando chamada pelo *receiver* é o inverso. Começa por esperar uma trama DISC. Após a receção desta, envia uma trama DISC e, de seguida, espera a receção de uma trama UA.

São utilizadas as funções **write\_message**, **read\_message** e **parseMessageType** de forma semelhante ao seu anterior uso.

# **Protocolo de aplicação**

O nível de aplicação divide-se em duas partes: a parte correspondente ao *sender* e a correspondente ao *receiver*. Cada uma destas partes tem uma função **main** que delineia a estrutura principal da aplicação correspondente.

* **main (sender.c)**

Esta função começa por chamar a função **setup**, que prepara a comunicação através de porta de série. De seguida, é chamada a função **llopen**, que pertence ao protocolo de comunicação, para estabelecer a comunicação com o *receiver*. Caso este estabelecer de comunicação for bem-sucedido, é extraída a informação do ficheiro a enviar através do uso da função **readFile**. Esta função abre o ficheiro passado como argumento de linha de comandos e retorna toda a informação nele presente na forma de um *array*. De seguida, são enviados um conjunto de pacotes de dados fazendo uso da função **llwrite**, pertencente ao protocolo de comunicação. É primeiro enviado o pacote START, de seguida todos os pacotes de dados pertencentes ao ficheiro, e finalmente o pacote END. Os pacotes START e END são criados pela função **controlPackage**, enquanto que os pacotes de dados são criados pela função **dataPackage**, que devolve a próxima secção do *array* de dados do ficheiro a enviar. Cada pacote de dados tem, no máximo, 260 *bytes* de informação. Finalmente, após o envio do pacote END, é chamada a função **llclose**, pertencente ao protocolo de comunicação, para terminar a ligação.

* **main (receiver.c)**

Esta função começa por, de modo análogo ao **main** de sender.c, chamar a função **setup**, que prepara a comunicação através de porta de série. De seguida, é chamada a função **llopen**, que pertence ao protocolo de comunicação, para estabelecer a comunicação com o *sender*. Caso este estabelecer de comunicação for bem-sucedido, é começada a receção dos pacotes enviados pelo *sender* fazendo uso da função **llread**, pertencente ao protocolo de comunicação. Logo a seguir à receção do primeiro pacote, é utilizada a função **parseMessageStart** para extrair informações como o nome do ficheiro a ser enviado e o seu tamanho. De seguida, após a receção de cada pacote é efetuada a chamada à função **parseMessageData**, que extrai a informação recebida no pacote, e **saveData**, que guarda essa informação num *array* que contém toda a informação recebida até agora, para posterior escrita da mesma em memória. Esta leitura de pacotes continua até a receção do pacote END. De seguida, é chamada a função **createFile**, que cria o ficheiro em memória e guarda nele toda a informação até agora recolhida no *array* referido anteriormente. Finalmente, é chamada a função **llclose**, pertence ao protocolo de comunicação, para terminar a ligação.

# **Validação**

Para forma a estudar a aplicação, foram efetuados os seguintes testes:

- Geração de curto circuito aquando do seu envio.

- Interrupção da ligação, no envio, por alguns milésimos de segundos.

- Envio de ficheiros de vários tamanhos.

Todos os testes concluídos na sua totalidade com sucesso.

# **Eficiência do protocolo de ligação de dados**

# **Conclusões**

# **Anexo I**

## **Makefile**

|  |  |
| --- | --- |
|  | **all: sender receiver** |
|  |  |
|  | **sender: sender.c** |
|  | **gcc -o sender -Wall sender.c protocol.c llopen.c llwrite.c llclose.c** |
|  | **receiver: receiver.c** |
|  | **gcc -o receiver -Wall receiver.c protocol.c llopen.c llread.c llclose.c -lm** |
|  | **clean :** |
|  | **rm sender receiver \** |

## **llclose.h**

|  |
| --- |
|  |
| **int llclose(int fd, int flag);**  **int llclose\_sender(int fd);**  **int llclose\_receiver(int fd);** |
|  |

## **llclose.c**

|  |  |
| --- | --- |
|  | **#include "llclose.h"** |
|  | **#include " protocol.h"** |
|  | **int llclose(int fd, int flag)** |
|  | **{** |
|  | **if(flag == RECEIVER)** |
|  | **return llclose\_receiver(fd);** |
|  | **else** |
|  | **return llclose\_sender(fd);** |
|  |  |
|  | **return -1;** |
|  | **}** |
|  |  |
|  | **int llclose\_receiver(int fd)** |
|  | **{** |
|  | **unsigned char BCC1 = A\_RECEIVER ^ C\_DISC;** |
|  | **unsigned char disc[6] = {FLAG, A\_RECEIVER, C\_DISC, BCC1, FLAG, '\0'};** |
|  |  |
|  | **unsigned char buf[255];** |
|  |  |
|  | **while(1)** |
|  | **{** |
|  | **if(read\_message(fd,buf) == 0) break;** |
|  | **}** |
|  |  |
|  | **if(parseMessageType(buf) == C\_DISC)** |
|  | **write\_message(fd,disc,5);** |
|  | **else** |
|  | **return -5;** |
|  |  |
|  |  |
|  | **while(1)** |
|  | **{** |
|  | **if(read\_message(fd,buf) == 0) break;** |
|  | **}** |
|  |  |
|  | **if(parseMessageType(buf) == C\_UA)** |
|  | **{** |
|  | **close(fd);** |
|  | **return 0;** |
|  | **}** |
|  |  |
|  | **return -6;** |
|  | **}** |
|  |  |
|  | **int llclose\_sender(int fd)** |
|  | **{** |
|  | **unsigned char BCC1 = A\_SENDER ^ C\_DISC;** |
|  | **unsigned char disc[6] = {FLAG, A\_SENDER, C\_DISC, BCC1, FLAG, '\0'};** |
|  |  |
|  | **write\_message(fd,disc,5);** |
|  |  |
|  | **unsigned char buf[255];** |
|  |  |
|  | **while(1)** |
|  | **{** |
|  | **if(read\_message(fd,buf) == 0) break;** |
|  | **}** |
|  |  |
|  | **BCC1 = A\_SENDER ^ C\_UA;** |
|  | **unsigned char ua[6] = {FLAG, A\_SENDER, C\_UA, BCC1, FLAG, '\0'};** |
|  |  |
|  | **if(parseMessageType(buf) == C\_DISC)** |
|  | **{** |
|  | **write\_message(fd,ua,5);** |
|  | **close(fd);** |
|  | **return 0;** |
|  | **}** |
|  |  |
|  | **return -5;** |
|  | **}** |
|  |  |

## **llopen.h**

|  |
| --- |
|  |
| **int llopen\_Receiver (int fd);**  **int llopen\_Sender(int fd);**  **int llopen(int fd, int flag);** |
|  |

## **llopen.c**

|  |
| --- |
| **#include "protocol.h" #include "llopen.h"** |
| **int llopen\_Receiver(int fd)** |
| **{** |
| **unsigned char BCC1 = A\_SENDER ^ C\_UA;** |
| **unsigned char ua[6] = {FLAG, A\_SENDER, C\_UA, BCC1, FLAG, '\0'};** |
|  |
| **disableAlarm();** |
|  |
| **unsigned char buf[255];** |
|  |
| **while(1)** |
| **{** |
| **if(read\_message(fd, buf) == 0) break;** |
| **}** |
|  |
| **// analisar sender info** |
| **if(parseMessageType(buf) == C\_SET)** |
| **{** |
| **write\_message(fd, ua, 5);** |
| **return 0;** |
| **}** |
| **else** |
| **return -5;** |
| **}** |
|  |
| **int llopen\_Sender(int fd)** |
| **{** |
| **(void) signal(SIGALRM, attend);** |
|  |
| **unsigned char BCC1 = A\_SENDER ^ C\_SET;** |
| **unsigned char set[6] = {FLAG, A\_SENDER, C\_SET, BCC1, FLAG, '\0'};** |
|  |
| **int cnt = 0;** |
| **unsigned char buf[255];** |
|  |
| **while(cnt < 3)** |
| **{** |
| **alarm(3);** |
| **disableAlarm();** |
|  |
| **write\_message(fd, set, 5);** |
|  |
| **if(read\_message(fd, buf) == 0) break;** |
|  |
| **cnt++;** |
| **}** |
|  |
| **if(cnt == 3)** |
| **return 2; //no confirmation recieved** |
|  |
| **// analisar receiver info** |
| **if(parseMessageType(buf) == C\_UA)** |
| **return 0;** |
| **else** |
| **return -6;** |
| **}** |
|  |
| **int llopen(int fd, int flag)** |
| **{** |
| **if(flag == SENDER)** |
| **return llopen\_Sender(fd);** |
| **else if (flag == RECEIVER)** |
| **return llopen\_Receiver(fd);** |
|  |
| **return -1;** |
| **}** |

## **llread.h**

|  |
| --- |
| **int llread(int fd, int flag, unsigned char\*\* message);** |
| **int checkBCC2(unsigned char \* package, int size);** |
|  |
| **unsigned char\* destuffing(unsigned char\* buf, int \*size);** |

## **llread.c**

|  |
| --- |
| **#include "llread.h"** |
| **#include "protocol.h"** |
| **int llread(int fd, int flag, unsigned char\*\* message)** |
| **{** |
| **unsigned char\* buf = malloc(600 \* sizeof(unsigned char));** |
|  |
| **while(1)** |
| **{** |
| **if(read\_message(fd, buf) == 0) break;** |
| **}** |
|  |
|  |
| **if(buf[2] != (unsigned char)(flag \* 64))** |
| **{** |
| **unsigned char c1;** |
| **if(flag == 0) c1 = C\_RR0;** |
| **else c1 = C\_RR1;** |
|  |
| **unsigned char BCC1 = A\_SENDER ^ c1;** |
| **unsigned char rr[6] = {FLAG, A\_SENDER, c1, BCC1, FLAG, '\0'};** |
|  |
| **write\_message(fd, rr, 5);** |
|  |
| **return -3;** |
| **}** |
|  |
| **if(buf[3] != (buf[1] ^ buf[2]))** |
| **return -4;** |
|  |
| **int size;** |
|  |
| **unsigned char\* destuffed = destuffing(buf + 4, &size);** |
|  |
| **if (checkBCC2(destuffed, size) == 1)** |
| **return -5;** |
|  |
| **\*message = destuffed;** |
|  |
| **unsigned char c1;** |
| **if(flag == 0) c1 = C\_RR1;** |
| **else c1 = C\_RR0;** |
|  |
| **unsigned char BCC1 = A\_SENDER ^ c1;** |
| **unsigned char rr[6] = {FLAG, A\_SENDER, c1, BCC1, FLAG, '\0'};** |
|  |
| **write\_message(fd, rr, 5);** |
|  |
| **return size;** |
| **}** |
|  |
| **int checkBCC2(unsigned char \* package, int size)** |
| **{** |
| **int i = 1;** |
| **unsigned char check = package[0];** |
|  |
| **for(; i < size - 2; i++)** |
| **check ^= package[i];** |
|  |
| **if(check == package[size - 2])** |
| **return 0;** |
| **else** |
| **return 1;** |
| **}** |
|  |
|  |
| **unsigned char\* destuffing(unsigned char\* buf, int \*size)** |
| **{** |
| **unsigned char\* destuff = malloc(600);** |
|  |
| **int i = 0, j = 0;** |
|  |
| **while(1)** |
| **{** |
| **if(buf[i] == 0x7E)** |
| **{** |
| **destuff[j] = buf[i];** |
| **break;** |
| **}** |
| **else if(buf[i] == 0x7D)** |
| **{** |
| **if(buf[i+1] == 0x5E)** |
| **destuff[j] = 0x7E;** |
| **else if(buf[i + 1] == 0x5D)** |
| **destuff[j] = 0x7D;** |
|  |
| **j++;** |
| **i+=2;** |
| **}** |
| **else** |
| **{** |
| **destuff[j] = buf[i];** |
| **j++;** |
| **i++;** |
| **}** |
| **}** |
|  |
| **(\*size) = j + 1;** |
|  |
| **return destuff;** |
| **}** |

## **llwrite.h**

|  |
| --- |
| **int llwrite(int fd, unsigned char\* package, int flag, int noPackage, FILE\* fileTimePackages);** |

## **llwrite.c**

|  |
| --- |
| **#include "llwrite.h"**  **#include "protocol.h"** |
|  |
| **int llwrite(int fd, unsigned char\* package, int flag, int noPackage, FILE\* fileTimePackages)** |
| **{** |
| **float start\_time = (float)clock() / CLOCKS\_PER\_SEC;** |
|  |
| **unsigned char BCC2;** |
| **if(package[0] == C2\_DATA)** |
| **BCC2 = calculateBCC2(package, 4 + package[2]\*256 + package[3]);** |
| **else** |
| **BCC2 = calculateBCC2(package, 5 + package[2] + package[2+package[2] + 2]);** |
|  |
| **int char\_count;** |
| **unsigned char \* stuff = stuffing(package, BCC2, &char\_count);** |
|  |
| **unsigned char\* message = heading(stuff, char\_count, flag);** |
|  |
| **int cnt = 0;** |
| **unsigned char buf[255];** |
|  |
| **while(cnt < 3)** |
| **{** |
| **alarm(3);** |
| **disableAlarm();** |
|  |
| **write\_message(fd, message, 6 + char\_count);** |
|  |
| **if(read\_message(fd, buf) == 0)** |
| **{** |
| **if((parseMessageType(buf) == C\_RR0 && flag == 1) || (parseMessageType(buf) == C\_RR1 && flag == 0))** |
| **{** |
| **float end\_time = (float)clock() / CLOCKS\_PER\_SEC;** |
| **fprintf(fileTimePackages, "%f\n", (end\_time - start\_time)\*1000);** |
|  |
| **noPackage != -1 ? printf("Success on sending package no.%d - Transfer time: %f seconds\n", noPackage, (end\_time - start\_time) \* 1000) :** |
| **printf("Success on sending Start package - Transfer time: %f seconds\n", (end\_time - start\_time) \* 1000);** |
| **break;** |
| **}** |
| **}** |
|  |
| **noPackage != -1 ? printf("Failure on sending package no.%d, try no.%d\n", noPackage,cnt + 1) :** |
| **printf("Failure on sending Start package, try no.%d\n", cnt + 1);** |
|  |
| **cnt++;** |
| **}** |
|  |
|  |
| **if(cnt == 3)** |
| **return 2; //no confirmation recieved** |
|  |
| **if(flag == 1)** |
| **return 0;** |
| **else** |
| **return 1;** |
| **}** |

## **protocol.h**

|  |
| --- |
| **#ifndef \_PROTOCOL\_H**  **#define \_PROTOCOL\_H** |
|  |
| **#include <sys/types.h>** |
| **#include <sys/stat.h>** |
| **#include <fcntl.h>** |
| **#include <termios.h>** |
| **#include <stdio.h>** |
| **#include <stdlib.h>** |
| **#include <unistd.h>** |
| **#include <string.h>** |
| **#include <signal.h>** |
| **#include <sys/stat.h>** |
| **#include <math.h>** |
| **#include <strings.h>** |
| **#include "time.h"** |
|  |
| **#define FLAG 0x7E** |
| **#define A\_SENDER 0x03** |
| **#define A\_RECEIVER 0x01** |
| **#define C\_SET 0x03** |
| **#define C\_DISC 0x0B** |
| **#define C\_UA 0x07** |
| **#define C\_RR0 0x05** |
| **#define C\_RR1 0x85** |
| **#define C\_REJ0 0x01** |
| **#define C\_REJ1 0x81** |
|  |
| **#define ERROR 0xFF** |
|  |
| **#define SENDER 0** |
| **#define RECEIVER 1** |
|  |
| **#define BEGIN 0** |
| **#define START\_MESSAGE 1** |
| **#define MESSAGE 2** |
| **#define END 3** |
|  |
| **#define C2\_START 0x02** |
| **#define C2\_DATA 0x01** |
| **#define C2\_END 0x03** |
| **#define T\_SIZE 0x00** |
| **#define T\_NAME 0x01** |
|  |
| **void attend();** |
|  |
| **void disableAlarm();** |
|  |
| **int read\_message(int fd, unsigned char buf[]);** |
|  |
| **void write\_message(int fd, unsigned char buf[], int size);** |
|  |
| **unsigned char parseMessageType(unsigned char buf[]);** |
|  |
| **unsigned char calculateBCC2(unsigned char \*message, int size);** |
|  |
| **unsigned char\* stuffing\_data\_package(const unsigned char\* package, const unsigned char BCC2, int\* char\_count);** |
|  |
| **unsigned char\* stuffing\_control\_package(const unsigned char\* package, const unsigned char BCC2, int\* char\_count);** |
|  |
| **unsigned char\* stuffing(const unsigned char\* package, const unsigned char BCC2, int\* char\_count);** |
|  |
| **unsigned char\* heading(unsigned char \* stuff, int count, int flag);** |
|  |
| **#endif** |

## **protocol.c**

|  |
| --- |
| **#include "protocol.h"** |
| **int alarm\_flag = 1;** |
| **int rejj = 0;** |
|  |
| **void attend()** |
| **{** |
| **alarm\_flag = 1;** |
| **}** |
|  |
| **void disableAlarm()** |
| **{** |
| **alarm\_flag = 0;** |
| **}** |
|  |
| **int read\_message(int fd, unsigned char buf[])** |
| **{** |
| **int state = BEGIN;** |
| **int pos = 0;** |
|  |
| **int res;** |
| **unsigned char c;** |
|  |
| **while(alarm\_flag != 1 && state != END)** |
| **{** |
| **res = read(fd,&c,1);** |
|  |
| **if(res > 0)** |
| **{** |
| **switch(state)** |
| **{** |
| **case BEGIN:** |
| **{** |
| **if(c == FLAG)** |
| **{** |
| **buf[pos] = c;** |
| **pos++;** |
| **state = START\_MESSAGE;** |
| **}** |
| **break;** |
| **}** |
| **case START\_MESSAGE:** |
| **{** |
| **if(c != FLAG)** |
| **{** |
| **buf[pos] = c;** |
| **pos++;** |
| **state = MESSAGE;** |
| **}** |
| **break;** |
| **}** |
| **case MESSAGE:** |
| **{** |
| **buf[pos] = c;** |
| **pos++;** |
| **if(c == FLAG)** |
| **state = END;** |
| **break;** |
| **}** |
| **default: state = END;** |
| **}** |
| **}** |
| **}** |
|  |
| **if(alarm\_flag == 1)** |
| **return 1;** |
|  |
| **return 0;** |
| **}** |
|  |
| **void write\_message(int fd, unsigned char buf[], int size)** |
| **{** |
| **write(fd,buf,size);** |
| **fflush(NULL);** |
| **}** |
|  |
| **unsigned char parseMessageType(unsigned char buf[])** |
| **{** |
| **if(buf[0] != FLAG)** |
| **return ERROR;** |
|  |
| **if(buf[1] != A\_SENDER && buf[1] != A\_RECEIVER)** |
| **return ERROR;** |
|  |
| **if((buf[2] ^ buf[1]) != buf[3])** |
| **return ERROR;** |
|  |
| **if(buf[2] == C\_DISC ||** |
| **buf[2] == C\_SET ||** |
| **buf[2] == C\_UA ||** |
| **buf[2] == C\_RR0 ||** |
| **buf[2] == C\_RR1 ||** |
| **buf[2] == C\_REJ0 ||** |
| **buf[2] == C\_REJ1)** |
| **{** |
| **if(buf[4] == FLAG)** |
| **return buf[2];** |
| **else** |
| **return ERROR;** |
| **}** |
|  |
| **return ERROR;** |
| **}** |
|  |
| **unsigned char calculateBCC2(unsigned char \*message, int size)** |
| **{** |
| **unsigned char bcc2 = message[0];** |
| **int i = 1;** |
|  |
| **for(; i < size; i++)** |
| **bcc2 ^= message[i];** |
|  |
| **return bcc2;** |
| **}** |
|  |
| **unsigned char\* stuffing\_data\_package(const unsigned char\* package, const unsigned char BCC2, int\* char\_count)** |
| **{** |
| **unsigned char\* stuff = (unsigned char \*)malloc(265 \* 2 \* sizeof(unsigned char));** |
| **\*char\_count = 1;** |
| **stuff[0] = package[0];** |
|  |
| **if(package[1] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[1] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[1];** |
|  |
| **stuff[(\*char\_count)++] = package[2];** |
|  |
| **if(package[3] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[3] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[3];** |
|  |
| **int count = 4;** |
| **int i = package[2] \* 256 + package[3];** |
|  |
| **for(; count < 4 + i; count++)** |
| **{** |
| **if(package[count] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[count] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[count];** |
| **}** |
|  |
| **if(BCC2 == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(BCC2 == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = BCC2;** |
|  |
| **return stuff;** |
| **}** |
|  |
| **unsigned char\* stuffing\_control\_package(const unsigned char\* package, const unsigned char BCC2, int\* char\_count)** |
| **{** |
| **int size = package[2];** |
|  |
| **unsigned char\* stuff = (unsigned char \*)malloc( (5 + size + package[3+size] \* 256 + package[4+size]) \* 2 \* sizeof(unsigned char) );** |
|  |
| **\*char\_count = 1;** |
|  |
| **stuff[0] = package[0];** |
|  |
| **if(package[1] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[1] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[1];** |
|  |
| **if(package[2] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[2] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[2];** |
|  |
| **int count = 3;** |
|  |
| **for(; count < (3+size); count++)** |
| **{** |
| **if(package[count] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[count] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[count];** |
| **}** |
|  |
| **if(package[3+size] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[3+size] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[3+size];** |
|  |
| **if(package[4+size] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[4+size] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[4+size];** |
|  |
| **count = 5 + size;** |
| **int start\_pnt = count;** |
| **size = package[4+size];** |
|  |
| **for(; count < (start\_pnt + size); count++)** |
| **{** |
| **if(package[count] == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(package[count] == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = package[count];** |
| **}** |
|  |
| **if(BCC2 == 0x7E)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5E;** |
| **}** |
| **else if(BCC2 == 0x7D)** |
| **{** |
| **stuff[(\*char\_count)++] = 0x7D;** |
| **stuff[(\*char\_count)++] = 0x5D;** |
| **}** |
| **else** |
| **stuff[(\*char\_count)++] = BCC2;** |
|  |
| **return stuff;** |
| **}** |
|  |
| **unsigned char\* stuffing(const unsigned char\* package, const unsigned char BCC2, int\* char\_count)** |
| **{** |
| **if(package[0] == C2\_DATA)** |
| **return stuffing\_data\_package(package, BCC2, char\_count);** |
| **else** |
| **return stuffing\_control\_package(package, BCC2, char\_count);** |
| **}** |
|  |
| **unsigned char\* heading(unsigned char \* stuff, int count, int flag)** |
| **{** |
| **unsigned char \* message = (unsigned char \*)malloc( (5 + count) \* sizeof(unsigned char));** |
|  |
| **message[0] = FLAG;** |
| **message[1] = A\_SENDER;** |
| **message[2] = (unsigned char)(flag \* 64);** |
| **message[3] = A\_SENDER ^ message[2];** |
|  |
| **int i = 4;** |
|  |
| **for(; i < 5 + count; i++)** |
| **message[i] = stuff[i - 4];** |
|  |
|  |
| **message[i] = FLAG;** |
|  |
| **return message;** |
| **}** |

## **receiver.c**

|  |
| --- |
| **#include "llopen.h"** |
| **#include "llread.h"** |
| **#include "llclose.h"**  **#include "protocol.h"** |
|  |
| **#define BAUDRATE B38400** |
| **#define MODEMDEVICE "/dev/ttyS1"** |
| **#define \_POSIX\_SOURCE 1 /\* POSIX compliant source \*/** |
| **#define FALSE 0** |
| **#define TRUE 1** |
|  |
| **volatile int STOP=FALSE;** |
|  |
| **int setup()** |
| **{** |
| **int fd;** |
| **struct termios oldtio,newtio;** |
|  |
| **/\*** |
| **Open serial port device for reading and writing and not as controlling tty** |
| **because we don't want to get killed if linenoise sends CTRL-C.** |
| **\*/** |
|  |
| **fd = open("/dev/ttyS0", O\_RDWR | O\_NOCTTY );** |
| **fflush(NULL);** |
|  |
| **if (fd <0) {perror("/dev/ttyS0"); exit(-1); }** |
|  |
| **if ( tcgetattr(fd,&oldtio) == -1) { /\* save current port settings \*/** |
| **perror("tcgetattr");** |
| **exit(-1);** |
| **}** |
|  |
| **bzero(&newtio, sizeof(newtio));** |
| **newtio.c\_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;** |
| **newtio.c\_iflag = IGNPAR;** |
| **newtio.c\_oflag = 0;** |
|  |
| **/\* set input mode (non-canonical, no echo,...) \*/** |
| **newtio.c\_lflag = 0;** |
|  |
| **newtio.c\_cc[VTIME] = 1; /\* inter-character timer unused \*/** |
| **newtio.c\_cc[VMIN] = 0; /\* blocking read until 5 chars received \*/** |
|  |
| **/\*** |
| **VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a** |
| **leitura do(s) pr�ximo(s) caracter(es)** |
| **\*/** |
|  |
| **tcflush(fd, TCIOFLUSH);** |
|  |
| **if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {** |
| **perror("tcsetattr");** |
| **exit(-1);** |
| **}** |
|  |
| **printf("New termios structure set\n");** |
|  |
| **return fd;** |
| **}** |
|  |
|  |
| **off\_t parseMessageStart(unsigned char\* message, unsigned char\*\* filename)** |
| **{** |
| **off\_t fileSize = 0;** |
|  |
| **int file\_size\_length = message[2];** |
|  |
| **int i = 1;** |
| **for(; i <= file\_size\_length; i++)** |
| **fileSize += message[2+i] \* pow(256,file\_size\_length - i);** |
|  |
| **int filename\_length = message[2+i+1];** |
|  |
| **unsigned char\* name = (unsigned char \*)malloc((filename\_length + 1)\* sizeof(unsigned char));** |
|  |
| **int k = i + 4;** |
| **int j = 0;** |
| **for(; k < filename\_length + i + 4; k++, j++)** |
| **name[j] = message[k];** |
|  |
| **name[j] = '\0';** |
|  |
| **(\*filename) = name;** |
|  |
| **printf("Number of Packages = %li\n", fileSize / 260 + 1);** |
| **printf("Filename = %s\n", \*filename);** |
|  |
| **return fileSize;** |
| **}** |
|  |
| **int parseMessageData(unsigned char\* message, int messageSize, unsigned char\*\* data)** |
| **{** |
| **int length = message[2] \* 256 + message[3];** |
|  |
| **unsigned char \* dataAux = malloc(length \* sizeof(unsigned char));** |
|  |
| **int i = 4;** |
| **for(; i < length + 4; i++)** |
| **dataAux[i-4] = message[i];** |
|  |
| **(\*data) = dataAux;** |
|  |
| **return length;** |
| **}** |
|  |
| **void saveData(unsigned char\* fileContent, unsigned char\* data, int sizeData, int \*index)** |
| **{** |
| **int i = 0;** |
| **for(; i < sizeData; i++)** |
| **fileContent[(\*index) + i] = data[i];** |
|  |
| **(\*index) += sizeData;** |
| **}** |
|  |
| **void createFile(unsigned char\* fileContent, unsigned char\* filename, off\_t size\_file)** |
| **{** |
| **FILE \*file = fopen((char\*)filename, "wb+");** |
| **fwrite(fileContent, 1, size\_file, file);** |
| **fclose(file);** |
| **}** |
|  |
|  |
| **int main(int argc, char\*\* argv)** |
| **{** |
| **int fd = setup();** |
|  |
| **if(llopen(fd, RECEIVER) == 0)** |
| **printf("Connected\n");** |
| **else** |
| **{** |
| **printf("Failed\n");** |
| **return -2;** |
| **}** |
|  |
| **// Initial flag for start package** |
| **int flag = 0;** |
| **unsigned char\* message;** |
| **int messageSize;** |
|  |
| **while((messageSize = llread(fd, flag, &message)) < 0);** |
| **flag = 1;** |
|  |
| **unsigned char\* filename;** |
| **off\_t size\_file = parseMessageStart(message,&filename);** |
|  |
| **unsigned char \* fileContent = malloc(size\_file \* sizeof(unsigned char));** |
|  |
| **int index = 0;** |
| **int counter = 0;** |
|  |
| **while(index < size\_file)** |
| **{** |
| **while((messageSize = llread(fd, flag, &message)) < 0);** |
| **(flag == 0) ? (flag = 1) : (flag = 0);** |
|  |
| **unsigned char \* data;** |
| **int sizeData = parseMessageData(message, messageSize, &data);** |
| **counter++;** |
|  |
| **saveData(fileContent, data, sizeData, &index);** |
|  |
| **printf("Received package no.%d\n",counter);** |
| **}** |
|  |
| **createFile(fileContent, filename, size\_file);** |
|  |
| **printf("Finished receiving file %s\n", filename);** |
|  |
| **return llclose(fd,RECEIVER);** |
| **}** |

## **sender.c**

|  |
| --- |
| **#include "llopen.h"** |
| **#include "llwrite.h"** |
| **#include "llclose.h"**  **#include "protocol.h"** |
|  |
| **#define BAUDRATE B38400** |
| **#define MODEMDEVICE "/dev/ttyS1"** |
| **#define \_POSIX\_SOURCE 1 /\* POSIX compliant source \*/** |
| **#define FALSE 0** |
| **#define TRUE 1** |
|  |
| **unsigned char n\_seq = 0;** |
| **volatile int STOP=FALSE;** |
|  |
| **int setup()** |
| **{** |
| **int fd;** |
| **struct termios oldtio,newtio;** |
|  |
| **/\*** |
| **Open serial port device for reading and writing and not as controlling tty** |
| **because we don't want to get killed if linenoise sends CTRL-C.** |
| **\*/** |
|  |
| **fd = open("/dev/ttyS0", O\_RDWR | O\_NOCTTY );** |
| **fflush(NULL);** |
|  |
| **if (fd <0) {perror("/dev/ttyS0"); exit(-1); }** |
|  |
| **if ( tcgetattr(fd,&oldtio) == -1) { /\* save current port settings \*/** |
| **perror("tcgetattr");** |
| **exit(-1);** |
| **}** |
|  |
| **bzero(&newtio, sizeof(newtio));** |
| **newtio.c\_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;** |
| **newtio.c\_iflag = IGNPAR;** |
| **newtio.c\_oflag = 0;** |
|  |
| **/\* set input mode (non-canonical, no echo,...) \*/** |
| **newtio.c\_lflag = 0;** |
|  |
| **newtio.c\_cc[VTIME] = 1; /\* inter-character timer unused \*/** |
| **newtio.c\_cc[VMIN] = 0; /\* blocking read until 5 chars received \*/** |
|  |
| **/\*** |
| **VTIME e VMIN devem ser alterados de forma a proteger com um temporizador a** |
| **leitura do(s) pr�ximo(s) caracter(es)** |
| **\*/** |
|  |
| **tcflush(fd, TCIOFLUSH);** |
|  |
| **if ( tcsetattr(fd,TCSANOW,&newtio) == -1) {** |
| **perror("tcsetattr");** |
| **exit(-1);** |
| **}** |
|  |
| **printf("New termios structure set\n");** |
|  |
| **return fd;** |
| **}** |
|  |
| **unsigned char \*readFile(unsigned char\* filename, off\_t \*sizeFile)** |
| **{** |
| **FILE \*file;** |
|  |
| **struct stat fileInfo;** |
| **unsigned char\* fileContent;** |
|  |
| **if( (file = fopen((char\*)filename, "rb")) == NULL)** |
| **{** |
| **perror("Error reading file.\n");** |
| **exit(-1);** |
| **}** |
|  |
| **stat((char\*)filename, &fileInfo);** |
| **(\*sizeFile) = fileInfo.st\_size;** |
|  |
| **fileContent = (unsigned char \*)malloc(fileInfo.st\_size);** |
|  |
| **fread(fileContent, sizeof(unsigned char), fileInfo.st\_size, file);** |
|  |
| **return fileContent;** |
| **}** |
|  |
| **unsigned char\* controlPackage(unsigned char c2, const unsigned char\* filename, const off\_t sizeFile)** |
| **{** |
| **int res = sizeFile / 256;** |
| **int quo = sizeFile % 256;** |
| **int count = 1;** |
|  |
| **while(res > 0)** |
| **{** |
| **res = quo / 256;** |
| **quo %= 256;** |
| **count++;** |
| **}** |
|  |
| **int size = (5 + strlen((char\*)filename) + count) \* sizeof(unsigned char);** |
| **unsigned char\* data = (unsigned char \*)malloc(size);** |
|  |
| **data[0] = c2;** |
| **data[1] = T\_SIZE;** |
| **data[2] = count;** |
|  |
| **res = sizeFile / 256;** |
| **quo = sizeFile % 256;** |
|  |
| **int i = 3;** |
|  |
| **if(res == 0) data[i] = quo;** |
| **else data[i] = res;** |
|  |
| **while(res > 0)** |
| **{** |
| **res = quo / 256;** |
| **quo %= 256;** |
| **i++;** |
| **if(res == 0) data[i] = quo;** |
| **else data[i] = res;** |
| **}** |
|  |
| **data[i+1] = T\_NAME;** |
| **data[i+2] = strlen((char\*)filename);** |
|  |
| **i+=3;** |
| **for(count = 0; count < strlen((char\*)filename); i++, count++)** |
| **data[i] = filename[count];** |
|  |
| **return data;** |
| **}** |
|  |
| **unsigned char\* dataPackage(unsigned char \* content, off\_t \*offset, off\_t end\_offset)** |
| **{** |
| **unsigned char\* package = malloc(264 \* sizeof(unsigned char));** |
|  |
| **package[0] = C2\_DATA;** |
|  |
| **package[1] = n\_seq % 255;** |
| **n\_seq++;** |
|  |
| **off\_t chars\_to\_send = end\_offset - \*offset;** |
|  |
| **if (end\_offset - \*offset > 260)** |
| **chars\_to\_send = 260;** |
|  |
| **if(chars\_to\_send > 255)** |
| **{** |
| **package[2] = 1;** |
| **package[3] = chars\_to\_send - 256;** |
| **}** |
| **else** |
| **{** |
| **package[2] = 0;** |
| **package[3] = chars\_to\_send;** |
| **}** |
|  |
| **int i = 0;** |
| **for(; i < chars\_to\_send; i++, (\*offset)++)** |
| **package[4+i] = content[\*offset];** |
|  |
| **return package;** |
| **}** |
|  |
| **int main(int argc, char\*\* argv)** |
| **{** |
| **if(argc != 2)** |
| **{** |
| **printf("Usage: %s <filename>\n", argv[0]);** |
| **return -1;** |
| **}** |
|  |
| **int fd = setup();** |
|  |
| **if(llopen(fd, SENDER) == 0)** |
| **printf("Connected\n");** |
| **else** |
| **{** |
| **printf("Failed\n");** |
| **return -2;** |
| **}** |
|  |
| **int fd\_time\_packages = open("packageTime.txt", O\_WRONLY | O\_APPEND | O\_CREAT, 0644);** |
| **int fd\_time\_taken = open("fileTime.txt", O\_WRONLY | O\_APPEND | O\_CREAT, 0644);** |
|  |
| **FILE \*fileTimeTaken = fdopen(fd\_time\_taken, "a");** |
| **FILE \*fileTimePackages = fdopen(fd\_time\_packages, "a");** |
|  |
| **//Opens the file to be sent** |
| **off\_t fileSize;** |
| **unsigned char\* fileContent;** |
| **fileContent = readFile((unsigned char \*)argv[1],&fileSize);** |
|  |
| **unsigned char\* start = controlPackage(C2\_START, (unsigned char \*)argv[1], fileSize);** |
| **off\_t offsetFile = 0;** |
|  |
| **float start\_time = (float)clock() / CLOCKS\_PER\_SEC;** |
|  |
| **if(llwrite(fd, start, 0,-1, fileTimePackages) == 2) //ERROR '-1' start package** |
| **return -1;** |
|  |
| **int flag = 1;** |
| **int counter = 1;** |
|  |
| **while(offsetFile != fileSize)** |
| **{** |
| **unsigned char\* package = dataPackage(fileContent, &offsetFile, fileSize);** |
|  |
| **flag = llwrite(fd, package,flag, counter,fileTimePackages);** |
| **counter++;** |
|  |
| **if(flag == 2)//ERROR** |
| **return -1;** |
| **}** |
|  |
| **float end\_time = (float)clock() / CLOCKS\_PER\_SEC;** |
|  |
| **printf("Finished to send file %s - Transfer time: %f seconds\n", argv[1], (end\_time - start\_time)\*1000);** |
|  |
| **fprintf(fileTimeTaken, "%f\n", (end\_time - start\_time)\*1000);** |
|  |
| **return llclose(fd,SENDER);** |
| **}** |

# **Anexo II**