

Protocolo de ligação de dados

**Redes de Computadores**

**Mestrado Integrado de Engenharia Informática e Computação**

10 de novembro de 2020

Daniel Garcia Silva, up201806524@fe.up.pt

Mariana Truta, up201806543@fe.up.pt

3ºano, Turma 3, Grupo 5

Índice

[Sumário 2](#_Toc55923916)

[Introdução 2](#_Toc55923917)

[Arquitetura e Estrutura do Código 3](#_Toc55923918)

[protocol 3](#_Toc55923919)

[stateMachines 4](#_Toc55923920)

[appSender 4](#_Toc55923921)

[appReceiver 4](#_Toc55923922)

[datalink 4](#_Toc55923923)

[Casos de uso principais 5](#_Toc55923924)

[Protocolo de Ligação Lógica 5](#_Toc55923925)

[int llwrite(int fd, char \*buffer, int length) 6](#_Toc55923926)

[int llread(int fd, char \*buffer) 6](#_Toc55923927)

[int llclose(int fd, int status) 6](#_Toc55923928)

[Protocolo da Aplicação 7](#_Toc55923929)

# Sumário

No âmbito da unidade curricular de *Redes de Computadores*, foi elaborado um projeto que consistia no desenvolvimento de um *software* que permitisse a **transferência** de ficheiros de um computador para o outro, estando estes ligados por um **cabo série**.

Ao longo deste relatório, será explicado como foram **cumpridos** todos os objetivos do projeto, tendo sido concluída uma aplicação **funcional** e sem **perdas de dados**.

# Introdução

Este primeiro projeto tinha dois grandes objetivos: implementar um **protocolo de ligação de dados**, especificado no guião fornecido pelos docentes, e testá-lo com uma **aplicação** simples de transferência de ficheiros. Relativamente ao ambiente de desenvolvimento, o trabalho foi realizado em LINUX, utilizando a linguagem de programação C e portas série RS-232 cuja comunicação é assíncrona.

Neste relatório, pretende-se tornar claro como foi possível a elaboração de um **serviço de comunicação fiável** entre dois computadores por via de uma porta de série assíncrona, apresentando detalhes de toda a **teoria** utilizada. Este relatório está estruturado da seguinte forma:

* **Arquitetura e estrutura de código:** descriçãodosblocos funcionais e interfaces implementadas e apresentação das *APIs*, principais estruturas de dados e funções;
* **Casos de uso principais:** identificação dos casos de uso e sequências de chamadas de funções;
* **Protocolo de ligação de dados:** identificação dos principais aspetos funcionais bem como a descrição da estratégia de implementação dos mesmos, sendo complementada com a apresentação de extratos de código;
* **Protocolo de aplicação:** identificação dos principais aspetos funcionais bem como a descrição da estratégia de implementação dos mesmos, sendo complementada com a apresentação de extratos de código;
* **Validação:** descrição dos testes efetuados com apresentação quantificada dos resultados;
* **Eficiência de protocolo de ligação de dados:** caraterização estatística da eficiência do protocolo, recorrendo a medidas sobre o código desenvolvido;
* **Conclusão:** síntese da informação apresentada nas secções anteriores; reflexão sobre os objetivos de aprendizagem alcançados.

# 

# Arquitetura e Estrutura do Código

O projeto está dividido em duas camadas muito bem definidas: a camada de **protocolo de ligação de dados** e a camada da **aplicação** (ver anexo).

O objetivo do **protocolo de ligação de dados** é fornecer um serviço de comunicação de dados **fiável** entre dois sistemas ligados por um **cabo série**. Esta camada contem, assim, todas as funções necessárias para a **abertura**, **fecho**, **escrita** e **leitura** da porta de série. Para além disso, tem também a seu encargo o controlo de **erros** e **fluxo** e o *stuffing*/*destuffing* de pacotes. Todas estas funcionalidades são asseguradas pelas seguintes funções:

## protocol

* ***alarmSenderHandler***/***alarmReceiverHandler*** – são chamadas quando o tempo do alarme do emissor ou do recetor, respetivamente, se esgota;
* ***SandWOpenClose*** - envia uma mensagem com o comando *send* e o endereço *sendAddress* e espera por uma resposta com o comando *receive* e o endereço *recAddress*, usando um mecanismo de *Stop&Wait;*
* ***sendOpenCloseFrame*** – cria e envia uma trama de acordo com o comando (*SET, DISC* ou *UA*) e o endereço passados como argumentos;
* ***sendAckFrame*** - cria e envia uma trama de rejeição ou de aceitação;
* ***receiveOpenCloseFrame*** - lê a mensagem recebida (*SET*, *DISC* ou *UA*) e chama a função *changeStateS* que irá interpretar o byte e mudar de estado de acordo com o mesmo;
* ***receiveAckFrame*** - lê uma trama ACK, chama a função que verifica se é válida ou não, *changeStateAck*, e retorna 1, caso tenha sido rejeitada, ou 0, se for válida;
* ***sendInfoFrame*** - cria e envia uma trama *I* com os dados em *info*, acrescentando a quantidade necessária de *0x00* até ser atingido o valor *IFRAME\_SIZE – 2* e garantindo assim que a *frame* tem sempre um tamanho fixo;
* ***receiveInfoFrame*** - é responsável por ler uma trama *I*, guardando o pacote de controlo ou de dados em *info*, após o *byte destuffing.* O seu valor de retorno depende se a trama foi ignorada, rejeitada ou aceite pela máquina de estados, *changeStateInfo,*ou se o tempo do alarme esgotou.

## stateMachines

* ***changeStateS*** - verifica se o byte recebido é o esperado e age de acordo com o mesmo, atualizando o estado atual da trama *SET*, *DISC* ou *UA*;
* ***changeStateInfo*** - verifica se o byte recebido é o esperado e age de acordo com o mesmo, atualizando o estado atual da trama *I*;
* ***changeStateAck*** - verifica se o byte recebido é o esperado e age de acordo com o mesmo, atualizando o estado atual da trama *REJ ou RR*.

Em relação à camada da **aplicação**, estaestá situada acima da camada de ligação de dados, sendo responsável pelo **envio** e **receção** de ficheiros. Tem de permitir enviar e identificar os **pacotes de controlo** bem como **dividir** o ficheiro em vários pacotes, no caso do emissor, ou **concatenar** e **interpretar** toda a informação recebida, no caso do recetor. Nesta camada, as funções principais são:

## appSender

* ***main*** -é responsável pela escrita de um ficheiro, sendo o *path* fornecido pelo utilizador.
* ***makeControlPacket*** – cria um pacote de controlo com o campo de controlo *START* ou *END* e com o tamanho e nome do ficheiro passados como argumentos;
* ***makeDataPacket*** - cria um pacote de dados com o número de sequência *N* e os dados *info*.

## appReceiver

* ***main*** – é responsável pela leitura de um ficheiro, sendo guardado em *“./images/ToReceive”*;
* ***parseInfo*** - encaminha o pacote de dados para função respetiva, tendo em conta

o seu primeiro byte (*START\_BYTE*, *DATA\_BYTE* e *END\_BYTE*);

* ***parseControlPacket*** - interpreta a informação guardada na *info*, armazenando o nome e o tamanho do ficheiro que irá ser recebido na *struct File*;
* ***checkControlPacket*** - verifica se o pacote de controlo de finalização contém a mesma informação que o pacote de controlo que sinalizou o inicio da transmissão, diferenciando apenas no campo de controlo;
* ***parseDataPacket*** - guarda os dados do ficheiro no campo *data* da *struct File*.

## datalink

Neste ficheiro, está contida a **interface Protocolo-Aplicação** que se resume essencialmente a quatro funções que serão descritas pormenorizadamente mais à frente: *llopen*, *llwrite*, *llread* e *llclose*. Estas funções facilitam a **comunicação** entre a camada da aplicação e a camada de ligação de dados, permitindo assim que a arquitetura cumpra o **principio de independência entre camadas**, isto é, nenhuma camada conhece os detalhes da outra.

Para facilitar o armazenamento de informação, foram utilizadas algumas **estruturas de dados definidas** em *dataStructures*: *enum* *State* e *enum* *AckState* que permitem guardar o estado atual da respetiva trama, *enum* *Command* que possibilita definir o tipo de comando e a *struct File* que armazena a informação necessária para a escrita do novo ficheiro. Para além disso, em *macros*, encontram-se todas as **macros** necessárias ao longo do projeto.

# Casos de uso principais

Inicialmente, é necessário **compilar** o programa, executando o comando *make* na pasta *src.* De seguida, é necessário executar *“./sender <porta> <path do ficheiro a enviar>”*, no caso do **emissor**, e *“./receiver <porta>”,* no caso do **recetor**. A porta de série tem de estar no formato *“/dev/ttySX”* sendo X o número da mesma.

A **transmissão**/**receção** dos dados é feita pela seguinte ordem:

|  |  |
| --- | --- |
| no caso do **emissor**,   1. **Estabelecimento** da ligação entre o emissor e recetor na função *llopen*; 2. **Criação** do pacote de controlo *START* e o seu **envio** com *llwrite*; 3. **Criação** de pacotes de dados com os dados do ficheiro e o seu **envio** em *llwrite*; 4. **Criação** do pacote de controlo *END* e o seu **envio** em *llwrite*; 5. **Terminação** da ligação entre o emissor e o recetor na função *llclose.* | no caso do **recetor**,   1. **Estabelecimento** da ligação entre o emissor e recetor na função *llopen*; 2. **Receção** e **análise** do pacote de controlo *START*, enviando uma **resposta**, com auxilio da função *llread*; 3. **Receção** e **análise** dos pacotes de dados, enviando uma **resposta**, com auxilio da função *llread*; 4. **Receção** e **análise** do pacote de controlo *END*, com auxilio da função *llread*; 5. **Escrita** dos dados recebidos num ficheiro com o mesmo nome do ficheiro enviado pelo emissor*;* 6. **Terminação** da ligação entre o emissor e o recetor na função *llclose.* |

# Protocolo de Ligação Lógica

O **protocolo de ligação lógica** é responsável por várias funcionalidades. Como a *API*  da porta de série tem ao seu encargo a **receção** dos dados da camada da aplicação e **estabelecimento** da ligação entre esta e a camada de ligação de dados, é conveniente explicar detalhadamente como é que as funções *llopen*, *llwrite*, *llread* e *llclose* se relacionam com o **protocolo da ligação lógica**. Estas utilizam o mecanismo de ***Stop&Wait***, isto é, envia-se um pacote de cada vez, esperando por uma resposta até se poder enviar outro. Se esta resposta não chegar, demorar a chegar (máximo 3 segundos) ou for uma resposta de erro, é iniciada uma nova **retransmissão** da mensagem enviada até ao limite de 3 retransmissões.

***int llopen(int port, int status)***

**Estabelece** a ligação entre o emissor e o recetor e **abre** a porta de série.

É responsável por **enviar** uma mensagem *SET* e **esperar** por uma mensagem *UA,* no caso do **emissor**. Esta ação é realizada pela função *SandWOpenClose* da camada de ligação de dados. Por sua vez, no caso do **recetor**, é chamada a função *receiveOpenCloseFrame* que irá **receber** a mensagem *SET* e posteriormente **envia** uma mensagem *UA* com o auxilio da função *sendOpenCloseFrame*.

## int llwrite(int fd, char \*buffer, int length)

Esta função **recebe** um buffer que será enviado para a função *sendInfoFrame*, onde será **criada** uma **trama de informação** com este *buffer*, após o *byte stuffing* do mesmo, e **enviada** para o recetor (ver anexo). De seguida, **espera** por uma mensagem *ACK*. Se o pacote tiver sido **rejeitado**, é **retransmitido**. Se tiver sido **aceite**, a transmissão do ficheiro **continuará**.

## int llread(int fd, char \*buffer)

Esta função chama *receiveInfoFrame* que irá **esperar** por uma trama I, analisando o conteúdo do seu cabeçalho à procura de qualquer erro. Posteriormente, é **enviado** um comando *REJ* ***/*** *RR* na função *sendAckFrame* caso a trama tenha sido **rejeitada** ou **aceite**, respetivamente. A trama só é rejeitada caso não seja um duplicado e tenha sido detetado um erro pelo respetivo *BCC*.

## int llclose(int fd, int status)

**Termina** a ligação entre o emissor e o recetor.

O emissor chama a função SandWOpenClose que **envia** o comando *DISC* e **espera** pela receção do comando *DISC* enviado pelo recetor. Termina com o **envio** do comando *UA* com o auxílio da função *sendOpenCloseFrame*. Por outro lado, o recetor, após **receber** o comando *DISC*, **envia** o comando *DISC* para o emissor na função *receiveOpenCloseFrame* e **aguarda** pela receção do comando *UA* com o auxilio da função *SandWOpenClose.* Em ambos os casos, é **fechada** a porta de série.

# Protocolo da Aplicação

O protocolo da aplicação tem a seu encargo os seguintes aspetos funcionais:

* **Envio de pacotes de controlo que sinalizam o inicio e o fim da transmissão pelo emissor**

Na *main* do ficheiro *appSender*, é chamada a função *makeControlPacket* (ver anexo) que **constrói** um pacote de controlo com o tamanho e o nome do ficheiro a enviar e são posteriormente enviados com o auxilio da função *llwrite*.

* **Envio de pacotes de dados contendo fragmentos dos dados do ficheiro a enviar pelo emissor**

Na *main* do ficheiro *appSender*, são lidos **fragmentos** com tamanho *MAX\_K* do ficheiro até se chegar ao fim do mesmo. A cada fragmento é acrescentado um cabeçalho em *makeDataPacket* (ver anexo) com o seu **tamanho** e o número de **sequencia** *sequenceN*, sendo posteriormente **enviado** com o auxilio de *llwrite*.

* **Leitura de pacotes de controlo e de dados pelo recetor**

Na *main* do ficheiro *appReceiver*, quando se **recebe** um pacote, é analisado o seu primeiro byte na função *parseInfo* e a informação recebida é passada à função respetiva de acordo com esse mesmo byte: no caso de ser o *START\_BYTE*, é chamada a função *parseControlPacket* (ver anexo) que é responsável por preencher os campos do **tamanho** e **nome** do ficheiro na *struct File* bem como guardar todo o pacote recebido no campo *controlPacket* para futura comparação com o pacote de terminação; no caso de ser o *DATA\_BYTE*, o pacote segue para a função *parseDataPacket* (ver anexo), onde serão guardados os **dados** no campo *data* da *struct File*; no caso de ser o END\_BYTE, é chamada a função *checkControlPacket* (ver anexo) que tem a seu cargo a **comparação** deste pacote com o pacote que sinalizou o inicio da transmissão. O número de sequência do pacote de dados é armazenado na variável estática *N*.

* **Criação do ficheiro recebido no recetor**

Na *main* do ficheiro *appReceiver*, após a receção de todos os pacotes, é realizada a **criação** e **escrita** do ficheiro recebido com o tamanho e nome enviados, sendo este guardado na pasta *“./imagesToReceive/”* (ver anexo).

# Validação

O programa foi capaz de transmitir uma imagem de 10 968 bytes e outra de 3 309 702 bytes, com e sem interrupção da porta de série, variando o BAUDRATE (entre 2400 e 38400) e o tamanho da trama I (entre 128 e 8192 bytes).

# Eficiência

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EFICIÊNCIA REAL | TAMANHO DA TRAMA I | | | | |
| BAUDRATE | 128 | 512 | 1024 | 2048 | 8192 |
| 2400 | 0,339295 | 0,362175 | 0,306896 | 0,28644 | 0,214731 |
| 4800 | 0,339229 | 0,362152 | 0,356341 | 0,286436 | 0,214728 |
| 9600 | 0,339065 | 0,3621 | 0,356309 | 0,329724 | 0,214719 |
| 19200 | 0,338853 | 0,362001 | 0,356232 | 0,329673 | 0,214705 |
| 38400 | 0,338265 | 0,361825 | 0,35613 | 0,329617 | 0,214686 |

Mediram-se os tempos, em segundos, de transmissão e de receção do total das tramas I, e usou-se o valor médio dos dois. Após variar o BAUDRATE e o tamanho (em bytes) da trama, dividiu-se o tamanho do ficheiro (pinguim.gif), em bits, pelo tempo medido, para se obter a velocidade em bits por segundo. Por fim, dividiu-se a velocidade medida pelo BAUDRATE, a velocidade teórica, e o resultado está na tabela a seguir.

# Conclusão

A implementação deste programa permitiu uma melhor aprendizagem dos conceitos que aplicámos, nomeadamente:

* O protocolo de ligação de dados, neste caso, Stop and Wait, bem como a noção de transparência, através do byte stuffing;
* A independência entre camadas, de ligação e de aplicação, e a sua interface, através da separação de processamento de pacotes e tramas.

A equipa gostaria ainda de referir que este trabalho prático apresentou algumas dificuldades, causadas principalmente pela dificuldade em testar o programa nos laboratórios.