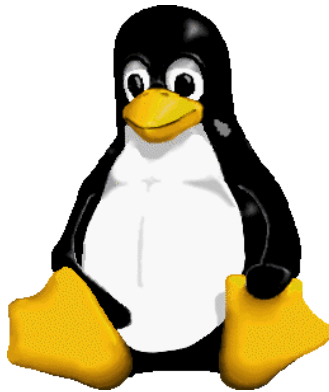


# **REDES DOS COMPUTADORES**

([L.EIC025](#))



## **PROJETO 1**

João Alves - up2018

Manuel Sá - up201805273

# Sumário

Este é o Primeiro trabalho de redes dos computadores no ano letivo 2021/2022. Este projeto visa explorar a transmissão de dados através de uma ligação de portas.

O objetivo deste relatório é abordar e esclarecer todos os métodos utilizados, dificuldades que ocorreram e conclusões que tiramos, durante a realização do trabalho.

## Introdução

O objetivo deste trabalho é implementar um protocolo de ligação de dados, através da transferência de dados. Será usada comunicação assíncrona com as portas série RS-232.

De seguida, poderemos analisar:

- A arquitetura do projeto;
- A estrutura do código;
- Os principais casos de uso;
- O protocolo de ligação lógica;
- O protocolo de ligação;
- A validação dos testes feitos;
- A eficiência;
- Conclusões finais.

Os pontos acima serão elaborados ao longo deste relatório, com base. É de notar o facto de alguns pontos serem mais explorados do que outros, devido a certos problemas que encontramos ao longo da realização do projeto.

## Arquitetura

O projeto foi feito em linguagem C. Existem ficheiros h, ficheiros c, um makefile e o ficheiro que vamos usar para ser transmitido. A lista completa de ficheiros é a seguinte:

- application.c
- application.h
- link.c
- link.h
- main.c
- makefile
- pinguim.gif
- statemachine.c
- statemachine.h

De seguida, será dada uma explicação mais detalhada para cada um destes ficheiros acima. Quando se corre o programa, caso a transferência tenha sido bem sucedida, um novo ficheiro é criado com o nome return\_file.gif.

# Estrutura do código

Esta seção visa elaborar com mais detalhe a função de cada ficheiro e o que cada um contém. Começamos pelo makefile.

O objetivo do makefile é compilar todos os ficheiros ou limpar executáveis e ficheiros desnecessários. O ficheiro pinguim.gif é o ficheiro que pretendemos transmitir através das portas. O ficheiro main.c tem como função reunir todas as informações necessárias que reúne das restantes. Vai coordenar e transmitir se tudo está a proceder de forma correta, assim como terminar o programa. Detecta se a porta que começou a correr é o emissor ou o receptor e com base nisso chama as devidas funções associadas a cada um. Também abre e lê o ficheiro de input (neste caso e como referido anteriormente, o pinguim.gif).

Passemos para os ficheiros application.c e application.h. A principal função deles é conter duas peças essenciais para o funcionamento do programa: Algumas variáveis globais importantes como o baudrate ou o escape octet, e funções como llopen, llwrite, llread ou llclose. Também contêm funções que criam packets e funções para os controlar.

Os ficheiros statemachine.c e statemachine.h possuem variáveis globais do recetor como a FLAG, A, C ou o BCC\_OK. Estas variáveis são usadas para verificar o estado do buffer. E é precisamente isso que faz a única função neste ficheiro. Verifica o estado no buffer e caso esteja em condições de o mudar muda-o.

Por último temos os ficheiros link.c e link.h. A principal função destes ficheiros é aglomerar variáveis e funções que são fundamentais para o funcionamento do protocolo. Variáveis como SET, DISC os diferentes tipos de A e C, RR entre outras como TRANSMITER ou EMISSOR. Quanto a funções que podemos encontrar aqui, é possível verificar funções para byte stuffing e byte destuffing, para a escrita e leitura de tramas i contendo os bytes transferidos, assim como funções de início e fim de ligação entre portas.

## Casos de uso

Existem dois casos de uso possíveis: execução como emissor e execução como recetor. Vejamos primeiro o caso em que somos o emissor. Para o programa executar como emissor, depois de fazer 'make all' é necessário colocar como input a porta e o nome do ficheiro de seguida. Ex: './protocol /dev/ttyS10 pinguim.gif'.

Após isto começará o processo de transferência. É feito o llopen e de seguida é enviado o control packet.

O llopen envia-nos para uma função chamada initiate\_connection. Esta começa por guardar as informações da porta e tenta enviar o SET através da função su\_frame\_write. De seguida espera por resposta com um compasso de espera entre alarmes de tentativas de obtenção de resposta. Não obtenha resposta, o programa para. Caso receba uma resposta, vai verificar o estado no buffer e caso este seja BCC\_OK, ele dá a ligação como estabelecida.

Depois de termos uma ligação estabelecida, e voltando à main, vai ser iniciada a transferência do ficheiro. Para isso, o programa vai chamar a função `llwrite`, que por sua vez irá abrir a função `i_frame_write`. Mais à frente será abordada com mais detalhe. Caso exista algum problema, a função pode tanto fechar o programa caso não consiga ler nenhuma trama `i`, como pode entrar em loop como nos aconteceu numa fase experimental. No caso de tudo correr como suposto, o programa terá enviado os bytes do ficheiro com sucesso para o recetor, recebendo uma confirmação do mesmo.

Passamos agora à fase final, onde na main é chamada a função `ffclose`. Isto leva-nos diretamente para a função `terminate_connection`. Esta função envia um DISC e como nos outros casos espera uma resposta. Caso não a receba retorna um erro e caso receba o estado `BCC_OK` do recetor, confirma uma transferência bem sucedida e retorna essa mesma informação. Na main fecha-se a ligação das portas e termina o programa.

Do lado do receptor, o processo é bastante semelhante no que toca à perceção de envios, respostas, tempos de espera, tentativas e retorno de respostas e erros. Todas as funções são usadas nos dois casos, apenas são direcionados para secções de código diferentes na função devido a uma flag que nos diz se é o emissor ou o receptor que está a chamar a função, o que permite receber e enviar respostas e tipos de tramas diferentes em comparação ao emissor. A única exceção estará na parte da transferência de dados, onde somos enviados para a função `read_i_frame` para recebermos os bytes enviados pelo emissor.

## Protocolo de ligação lógica/Aplicação

Aqui estão algumas estratégias utilizadas no código, no que toca à implementação de algumas funcionalidades necessárias.

- `Alarm( )` - O uso da função `alarm` é crucial para o retorno de erros. Esta função está diretamente relacionada com outra chamada `sig_handler`. A função `sig_handler` determina o que é feito no fim de cada alarm. No nosso caso faz o seguinte:

```
void sig_handler(int signum){
    alarmFlag=1;
    if(alarmCount == numTransmissions){
        return;
    }
    printf("Alarm %d\n\n\n", alarmCount);
    alarmCount++;
    alarm(timeout);
}
```

Sempre que o tempo do alarm acaba, uma flag é ativada para dizer que ultrapassou o tempo limite daquela tentativa. Caso seja o último alarm, ela retorna para a função que tem o alarm perceber que atingiu o limite de tentativas. Caso contrário, incrementa a variável que guarda esse número de tentativas e faz com que comece um novo alarm.

- Byte stuffing e byte destuffing: Funções que usamos para remover e colocar de novo bytes como FLAGS ou escape octets do packet, pois podem afetar a leitura do mesmo.

```
unsigned char * byte_stuffing(unsigned char *packet, int *length){
    //so do data no i packet
    unsigned char *stuffed_packet = NULL;
    stuffed_packet = (unsigned char *)malloc( *length * 2);
    int j = 0;
    for(int i = 0; i < *length; i++){
        if(packet[i] == FLAG){
            stuffed_packet[j] = 0x7d;
            stuffed_packet[++j] = 0x5e;
        }
        else if(packet[i] == ESCAPE_OCTET){
            stuffed_packet[j] = 0x7d;
            stuffed_packet[++j] = 0x5d;
        }
        else stuffed_packet[j] = packet[i];

        j++;
    }
    //printf("data = %x, stuffed data = %x\n", packet[i], stuffed_packet[i] );
    *length = j;
    return stuffed_packet;
}
```

```
unsigned char * byte_destuffing(unsigned char *packet, int *length){
    //so do data no i packet
    unsigned char *destuffed_packet = NULL;
    destuffed_packet = (unsigned char *)malloc(*length * 2);
    int j = 0;
    for (int i = 0; i < *length; i++) {
        if (packet[i] == ESCAPE_OCTET)
            destuffed_packet[j] = packet[++i] ^ 0x20;
        else
            destuffed_packet[j] = packet[i];

        j++;
    }
    *length = j;
    return destuffed_packet;
}
```

Como é possível observar, no código procuramos endereços que são flags ou escape octets e colocamos numa string à parte. Quando o packet acaba de ser lido, são de novo colocados os valores removidos na função byte\_stuffing.

# Validação

No que toca à validação, não foram feitos testes suficientes para testar muitos casos que diferenciasssem uns dos outros. Foram feitos quatro tipos de teste: envio de teste: envio normal do pinguim.gif em portas virtuais e portas físicas, e envio sem recetor de pinguim.gif em portas virtuais e físicas.

Nos dois primeiros casos, o programa enviou com sucesso o ficheiro. Todas as mensagens e mensagens foram bem enviadas e recebidas, bem como todos os bytes do ficheiro. As imagens abaixo mostram o output do emissor e do recetor respetivamente, usando portas virtuais:

```
manel@derpro:~/Desktop/feup/rc/RC$ ./protocol /dev/ttyS11 pinguim.gif
New termios structure set
UA from SET message recieved
main- total file bytes = 10968
length in main is 10968
starting transmission of CONTROL packet
frame length = 55RR from i message recieved
starting transmission of I packet
frame length = 11043RR from i message recieved
starting transmission of last CONTROL packet
frame length = 11RR from i message recieved
terminate connection(transmitter) starting
DISC from DISC message recieved
manel@derpro:~/Desktop/feup/rc/RC$
```

```
manel@derpro:~/Desktop/feup/rc/RC$ ./protocol /dev/ttyS10
New termios structure set
establish connection - SET recieved!
after ua received
receiver reading first control packet
received final flag!
data packet received!
receiver received packet!
data size is 50
receiver reading first data packet
Received start
received final flag!
data packet received!
receiver received packet!
data size is 10970
receiver reading last control packet
received final flag!
data packet received!
receiver received packet!
data size is 6
terminate connection(receiver) starting
DISC recieved!
UA recieved!
SIZE OF READ IS 10970
```

```
manel@derpro:~/Desktop/feup/rc/RC$
```

Como é possível observar todas as tramas e bytes são devidamente passados, obtendo um novo ficheiro no directório, o gif que foi transferido.

No entanto, inicialmente não conseguíamos fazer a transferência em portas físicas porque entramos num loop ao enviar tramas i. Viemos a descobrir que o emissor não dava tempo suficiente para o recetor responder de volta devido à linha:

```
(fcntl(fd, F_SETFL, VTIME);
```

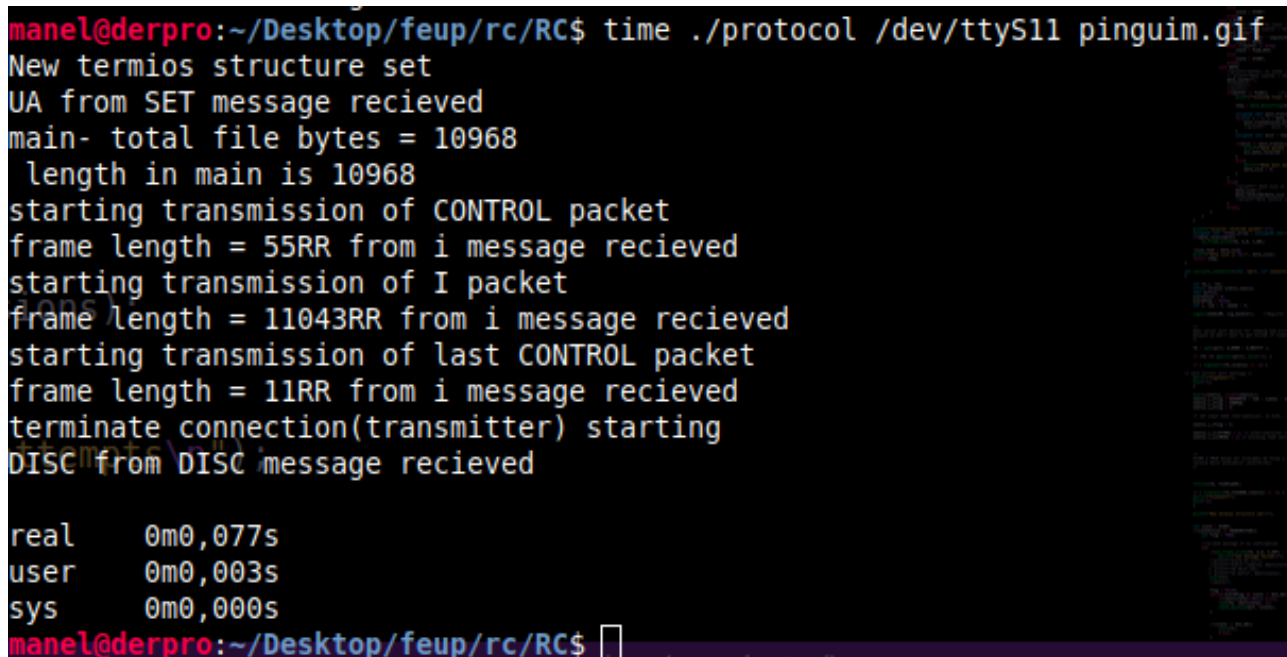
A transferência física demora mais tempo em comparação às portas virtuais, por isso é que o programa funcionava em portas virtuais e não em portas físicas.

Nos casos em que executamos apenas o lado emissor, apesar deste conseguir dar bem os alarmes no número correto de tentativas e no correto intervalo de tempo, o programa ficava preso na função read e não nos retornava o print de erro por não chegar lá.

## Eficiência

No que toca à eficiência, este conjunto de ficheiros encontra-se relativamente bem organizado e apresenta boas condutas para um código eficiente. Recorremos com bastante frequência ao uso de variáveis globais, que até nos permitem usar as mesmas funções para os diferentes casos de emissão e recepção.

Em condições favoráveis (neste caso será uma transferência do gif numa ligação estável de portas virtuais) o output é apresentado em 0,077 segundos, como mostra na imagem.



```
manel@derpro:~/Desktop/feup/rc/RC$ time ./protocol /dev/ttyS11 pinguim.gif
New termios structure set
UA from SET message recieved
main- total file bytes = 10968
length in main is 10968
starting transmission of CONTROL packet
frame length = 55RR from i message recieved
starting transmission of I packet
frame length = 11043RR from i message recieved
starting transmission of last CONTROL packet
frame length = 11RR from i message recieved
terminate connection(transmitter) starting
DISC from DISC message recieved

real    0m0,077s
user    0m0,003s
sys     0m0,000s
manel@derpro:~/Desktop/feup/rc/RC$
```

# Conclusões

Tendo em conta tudo o que foi abordado em cima, é possível tirar deste projeto algumas conclusões finais.

O trabalho apresentado tem aspetos a melhorar. Alguma limpeza de código e correção de bugs deveria ser feita. Nos casos que o favorecem, o programa apresenta bons resultados num bom intervalo de tempo.

Para além disto tudo, pode-se afirmar que este projeto visa introduzir-nos ao conceito de ligações de rede e dar-nos o head start para nos lançarmos na área. Foi-nos útil para perceber certos conceitos que nos serão valiosos num futuro próximo.