FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Labwork 1 –
Serial Port Communication

Diogo Landau, Gonçalo Ferreira
MIEEC – RCOM

2020

Índice

Sumário

No âmbito da unidade curricular de Redes de Computadores, no curso Mestrado integrado de Engenharia Eletrotécnica e Computadores, foi-nos proposto o desenvolvimento de uma camada de ligação de dados que permitisse duas aplicações a correr em computadores distintos conseguissem enviar e receber ficheiros.

O presente relatório permitiu desenvolver conhecimento quanto à ligação porta série, e ao mesmo tempo os diferentes cuidados que um protocolo de comunicação deve ter para a correta receção dos dados na presença de erros.

Introdução

O objetivo do trabalho realizado é o de implementar um protocolo de ligação de dados que faz recurso à porta série, e que tenha a capacidade de fornecer uma comunicação resiliente com o objetivo de transferir informação entre um Transmissor e Recetor. Quanto ao relatório, este visa documentar as decisões tomadas com a devida justificação, tendo a seguinte estrutura:

- **Arquitetura e Estrutura do Código:**
  Contém informação sobre: as APIs desenvolvidas para comunicar com a camada da aplicação; As principais estruturas de dados; As principais funções e como estas interagem com a arquitetura.
- **Casos de uso principais**
  Identificação das mesmas, e a sequência de chamada das funções.
- **Protocolo de ligação de dados**
  Identifica os principais aspetos funcionais e uma explicação quanto à estratégia de implementação pela qual se optou.
- **Protocolo de aplicação**
  Descreve o funcionamento da aplicação fornecida pelos docentes, e como moldou a estratégia de implementação do presente trabalho.
- **Validação**
  Descrição dos testes realizados com apresentação quantificada dos resultados se possível
- **Elementos de Valorização**
  Identificação dos elementos de valorização implementados e uma descrição da estratégia de implementação;
- **Conclusões**
  síntese da informação apresentada nas secções anteriores e uma reflexão sobre os objetivos de aprendizagem alcançados

Arquitetura e Estrutura do Código

2.1. **Arquitetura**

O código foi estruturado por forma a tentar otimizar tanto as funções como as variáveis partilhadas dentro de cada processo. A aplicação inclui o ficheiro *linklayer.h*, que por sua vez declara as funções que se encontram dentro dos 3 ficheiros de objetos *shared*, *writer* e *reader*. Estes três ficheiros contêm o grupo de funções responsáveis pela ligação dos dados.

Devido às diversas funções internas que se foram desenvolvendo ao longo do projeto, criou-se um ficheiro de objetos *funcs* que inclui todas estas, a serem utilizadas pelas funções do protocolo de ligação de dados.

Pela natureza do programa realizado, as funções *llwrite* e *llread* são executadas exclusivamente por dispositivos diferentes, pelo que foram implementadas em ficheiros de objetos diferentes também, *writer* e *reader* respetivamente.

2.1.1. **Funções e Variáveis Partilhadas**

No momento da chamada da função *llopen*, as variáveis externas *linkRole* e *fd* são definidas pelo ficheiro de objetos *shared*, que permite às funções do protocolo de ligações a enquadrarem-se no seu contexto no processo.

Já as funções *llopen* e *llclose* são independentes do papel do processo e são partilhadas entre o transmissor e recetor através do *shared*. A execução destas funções é que depende do contexto.

2.2. **Estrutura do Código**

- *Main*, fornecida pelo professor, abre as ligações em cada terminal:
  - Com o argumento "TX", inicia a aplicação em modo de transmissão;
  - É criada uma *struct* 'll' que guarda as informações de função, *baudrate*, número de tentativas de reconexão, e a duração de um alarme;
  - Abre o ficheiro a ser enviado;
  - Corre um ciclo contínuo onde são enviados *payloads* de um certo tamanho especificado para a camada link;
  - O buffer que armazena *payloads* também terá um byte a sinalizar o estado da transmissão do ficheiro na posição 0 (se estiver a 0, termina o envio do ficheiro).

- *Identifiers*.h – ficheiro *header* que armazena os valores dos bytes dos cabeçalhos das tramas.

- *State*.h – ficheiro *header* com os valores atribuídos para os vários estados possíveis na máquina de estados.

- *Linklayer*.h – é um ficheiro *header*, fornecido pelos docentes, com as especificações e argumentos das funções a implementar na camada de link. É aqui onde estão as declarações de *llopen*, *llclose*, *llread* e *llwrite*, a *struct 'linkLayer'*, e outras constantes úteis. Na compilação do programa, o *linker* recebe os *object* files do nosso código:

- *Shared*.c – ficheiro de código onde estão implementadas as funções partilhadas entre ambos *transmitter* e *receiver*, nomeadamente *llclose* e *llopen*. Estas funções servem para iniciar e terminar a ligação na camada de ligação.

- *Funcs*.c – neste ficheiro estão implementadas funções internas, que serão explicadas posteriormente.

- *Reader*.c – ficheiro de código onde são analisados os pacotes recebidos, e enviados pacotes de resposta, de acordo com as especificações que nos foram dadas.

- *Writer*.c – ficheiro de código que inicializa o alarme, e constrói o pacote a ser enviado, com várias funções para preparar o *header*, ler os dados, fazer o *stuffing*, etc..

Passaremos agora a explicar o funcionamento das principais funções do programa.

```
unsigned char
receiveFrame(int fd, unsigned char **message, int *message_size)
```

Esta função lê um byte de cada vez da trama recebida e recorre a uma máquina de estados para ler e interpretar esse byte. O valor retornado é o byte de comando lido. Esta máquina de estados está esquematizada a seguir:

Estado inicial: START:

- Transita para: FLAG_RCV se o primeiro byte é uma flag.

FLAG_RCV:

- Verifica o próximo byte e transita para:
- A_RCV e armazena o endereço em `header[0]` se o byte seguinte for o address.
- FLAG_RCV se o byte for uma flag.
- START para qualquer outra situação.

A_RCV:

- Verifica o próximo byte e transita para:
- C_RCV e armazena o comando em `header[1]` se o byte seguinte for um comando.
- FLAG_RCV se o byte for uma flag.
- START para qualquer outra situação.

C_RCV:

- Verifica se o próximo byte é um BCC e transita para:
- BCC_OK se o BCC estiver de acordo com o header recebido, verificando `header[0]` e `header[1]`.
- FLAG_RCV se o byte for uma flag.
- START para qualquer outra situação.

BCC_OK:

- Verifica o próximo byte e transita para:
- STOP caso o byte lido corresponde à FLAG
- BCC_OK caso contrário.

Caso se tenha recebido a última FLAG, termina-se a máquina de estados, e retornam-se os dados caso existam.

```
int BCC2_check(unsigned char *data, int size, unsigned char BCC)
```

Lê um vetor de dados e verifica se o BCC2 está correto.

```
unsigned char
*stuff(unsigned char *original, int og_size, int *stuffed_size)
```

Lê um vetor "*original" que contém os bytes sem stuffing, e retorna o apontador do vetor dos bytes com stuffing. No inicio da função, aloca um novo vetor de tamanho 2*tamanho original (o máximo teórico possível), verifica e compara cada byte do vetor com os valores de FLAG ou ESCAPE, e muda os bytes. No fim, reajusta o tamanho do vetor de acordo com o número de bytes que foram alterados.

```
int
de_stuff(unsigned char *stuffed, int stuffed_size, unsigned char *origina
l, int *og_size, unsigned char *BCC)
```

É o inverso da função anterior.

```
unsigned char *build_command_header(unsigned char C, int role)
```

Cria e retorna um vetor com o cabeçalho de comando de uma trama, de acordo com a função especificada (nomeadamente, a escolha de endereço depende da função).

```
void alarm_handler() e void initialize_alarm()
```

Funções responsáveis por inicializar e controlar os alarmes, utilizando o *interrupt* especificado (SIGALARM).

```
int llopen(linkLayer connectionParameters)
```

Abre uma ligação nova assíncrona de acordo com as especificações e o código base fornecidos. Para além disso, inicializa o alarme para o transmissor, e cria o *2-way handshake* inicial, sempre respeitando o limite de tentativas disponível para retransmissões.

```
int llclose(int showStatistics)
```

Termina a ligação com um 3-way handshake final, e liberta a memória alocada pelos apontadores e os descritores de ficheiro. Para além disso, implementa a funcionalidade de exibição de estatísticas da conexão.

```
int llread(char *packet)
```

Recebe uma trama (executando `receiveFrame()`) verifica o byte de controlo, faz destuffing, verifica o BCC2, e se tudo estiver correto, envia uma resposta positiva. Senão, envia

uma resposta negativa, ou envia um ACK para o pacote anterior caso seja uma trama repetida. Para além disso, verifica se houve receção de handshake inicial devido ao caso de re-estabelecimento da conexão.

```
int llwrite(char *buffer, int buffer_size)
```

Inicializa um alarme e cria uma trama nova, através da função `*build_command_header.` De seguida, cria um campo BCC e faz stuffing aos dados. No fim de tudo, junta estas porções da trama numa só, terminando com um FLAG no fim. No final, envia a trama, tendo em conta os limites de retransmissões e o estado do alarme, e espera pela receção de uma trama ACK ou NACK. No caso de se exceder o limite de retransmissão, tenta-se estabelecer a ligação novamente.

Casos de Uso Principais

Os casos de uso principais são: a interface através da linha de comandos, onde o utilizador seleciona que função o seu dispositivo deve ter, que porta utilizará para a comunicação, e que ficheiro quer enviar; e a transmissão do ficheiro através da porta série, sendo que há a possibilidade de visualizar estatísticas do desempenho da transferência do mesmo.

O utilizador deverá então introduzir na linha de comandos o nome do programa, e de seguida, como argumentos, a porta a ser utilizada o papel do processo, rx ou tx, e o ficheiro que pretende enviar (caso seja o transmissor).

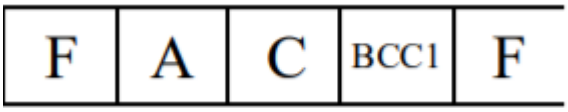
A sequência das funções executadas é:

- Transmissor escolhe o ficheiro a enviar;
- Estabelece-se uma ligação entre os dois computadores;
- Se a ligação se estabelecer com sucesso, inicia-se a transmissão do ficheiro;
- Se a transmissão ocorrer com sucesso, o recetor recebe os dados;
- Os dados são armazenados num ficheiro exatamente igual ao original;
- Término da ligação.

Protocolo de Ligação Lógica

O protocolo implementa uma ligação e transmissão assíncrona em série entre dois computadores. Para a comunicação entre eles, enviam-se tramas que podem ser de 3 tipos: tramas de supervisão, tramas não-numeradas e tramas de informação.

As tramas de supervisão e não numeradas são tramas de controlo, por não conterem dados do ficheiro a ser enviado, e servem apenas para sinalizar e confirmar ações, ou mudar o estado da transmissão. Portanto seguem uma estrutura igual à exemplificada abaixo.



O primeiro e último bytes são “F”, de FLAG, e delimitam a trama.

Os três bytes seguintes fazem parte do “header”, cabeçalho de uma trama, e têm a mesma estrutura nas tramas dos três tipos:

O segundo byte é “A” de ADDRESS, e é: 0x03 para comandos do transmissor e respostas do recetor, e 0x01 para comandos do recetor e respostas do transmissor.

O terceiro byte é “C” de COMMAND, e determina o tipo de trama de comando que se trata.

O quarto byte é “BCC1” para Block Check Character 1, e serve como byte de verificação de erros do cabeçalho.

As tramas de informação também estão delimitadas com uma flag, e possuem também um cabeçalho com a mesma estrutura anteriormente descrita. A diferença resume-se no facto de possuírem bytes de dados a seguir ao BCC1, seguido de um outro BCC para os dados, “BCC2”.

O recetor executa o programa em modo “rx” e o transmissor executa o programa em modo “tx”. No início, executa-se a função llopen, onde é iniciada a comunicação. A ligação entre ambos os computadores inicia-se com um 2-way handshake, onde o transmissor envia SET e o recetor responde com um UA. O transmissor também deve inicializar já temporizadores (**ver código em anexo, “shared.c”, linhas 70 a 95).**

De seguida, inicia-se a transmissão dos pacotes com dados. Na nossa implementação, o link layer recebe os payloads da camada de aplicação e envia-os como tramas de informação **(ver “writer.c”, linhas 14 a 72).** O recetor confirma a receção dos pacotes de dados enviando uma trama de supervisão com o comando “RR” de volta. Os dados são guardados num buffer e entregues à camada de aplicação. **(ver “reader.c”, linhas 8 a 62).**

Por fim, a desconexão efetua-se depois do envio da ultima trama e receção da respetiva confirmação, com um 3-way handshake: envia-se uma trama DISC, o recetor envia de volta uma trama DISC, e o transmissor termina enviando um UA **(ver “shared.c”, linhas 99 a 169).**

A camada de ligação implementa também um sistema de retransmissão de tramas, onde um temporizador é ativado aquando do envio de informação, e dispara um alarme caso TIMEOUT_DEFAULT segundos passem sem receber a trama de confirmação válida. Excedendo MAX_RETRANSMISSIONS_DEFAULT disparos do alarme, efetua-se uma reconexão e repete-se todo o processo. Se apenas os dados da trama forem inválidos, basta o recetor enviar uma

trama REJ e o transmissor reenvia a última trama, não havendo necessidade de esperar até que o alarme dispare.

Validação e Elementos de Valorização

Foram efetuados testes em computadores com uma ligação série física, e no próprio computador, com dois processos e uma ligação virtual a correr.

Implementou-se um script onde o utilizador pode configurar o *baudrate*, tamanho máximo dos *payloads*, número de retransmissões, probabilidade de erro e o intervalo de *timeout*. O programa correu sem problemas em todos os ambientes de teste. Os ficheiros responsáveis por implementar o código descrito encontra-se no anexo no diretório "projeto_1/scripts" e ficheiro "rcom1/build.sh".

A probabilidade de erro tem o propósito de decidir se se deve gerar erro aleatório sempre que se recebe um byte. Caso a decisão seja positiva, gera-se o erro numa posição aleatória do byte. Este código é descrito nas funções "random_error" e "random_position" do ficheiro "projeto_1/src/funcs.c".

Com a ligação física, pôde-se desativar temporariamente a ligação através do botão ON/OFF, e introduzir ruído nos pinos da ficha. O programa foi resistente às duas fontes de erros.

A ligação virtual foi implementada recorrendo ao comando *socat*, e mais tarde ao ficheiro cable fornecido pelo professor. Foi criado um ficheiro script onde se pode configurar e correr a transmissão dos dados e o gerador de erros de forma mais simples no ficheiro "rcom1/application/test_script".

O programa também foi testado com vários tamanhos, *payloads*, *baudrates*, *timeouts*, probabilidades de erro e valor máximo de tentativas, e correu como esperado.

O programa apresenta muito bom desempenho, graças ao número reduzido de operações na memória *heap* – apenas 1 por cada trama recebida, 2 por cada byte *stuffing*, e à ausência total de *memory leaks*.

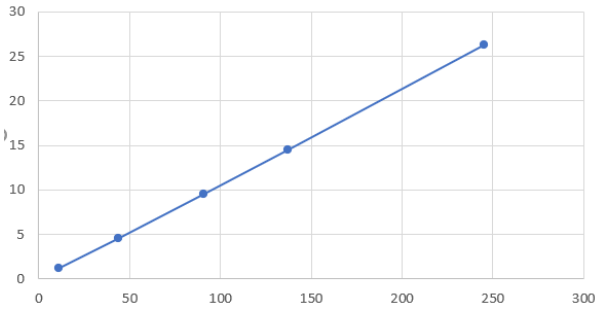


Figura 1 - Tempo de transmissão para imagens diferentes, payload 1000, p_error 0

O gráfico demonstrado representa a duração da transmissão dos dados em segundos. O tamanho dos ficheiros em KBytes são 11kb, 44,1kb, 90,6kb, 137kb, 245kb. As condições de envio são sempre as mesmas, tendo escolhido a opção *default* do ficheiro "rcom1/application/build.sh". Como esperado o tempo de envio cresce linearmente com o tamanho do ficheiro.

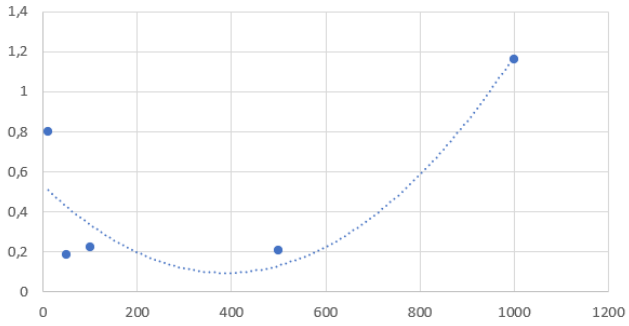


Figura 2 - tempo de envio em segundos em relação ao tamanho do payload

Este gráfico representa o tempo de envio em segundo no eixo dos y, e o tamanho do *payload* no eixo dos x. Foram testados tamanhos de 10, 50, 100, 500 e 1000 bytes. O ótimo tamanho do *payload* encontra-se entre os 500 bytes. Quanto mais reduzido for o *payload* deste valor, maior é a relação (*header size)/(payload size)* sendo necessário então enviar mais bytes para conseguir enviar a mesma informação. No caso de aumentarmos o tamanho de *payload* relativo ao ótimo, verifica-se que o tempo de envio cresce, provavelmente devido aos acessos à memória por parte do computador serem menos eficientes, devido ao tamanho da trama.

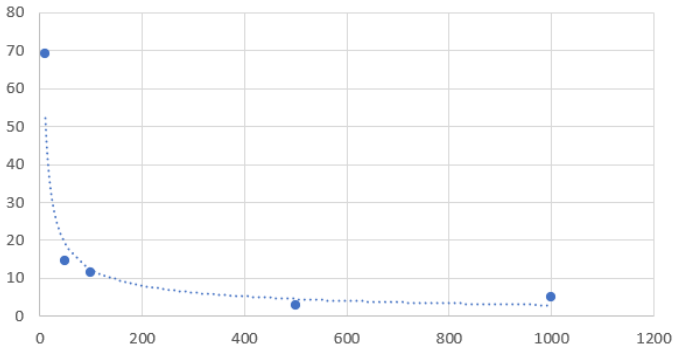


Figura 3 - tempo de envio para diferentes payloads com p_error 0.1

Para uma ligação com probabilidade de erro de 0.1%, podemos observar que o tempo que leva ao envio de informação decresce com o tamanho da trama. Apesar deste resultado, pode-se afirmar que esta informação induz em erro pois o ficheiro final no caso de um payload maior, está sempre com mais erros relativo ao original. Isto deve-se ao facto de se utilizar apenas um byte de verificação da paridade de todos os bits na secção de informação. Assim, caso exista um erro na mesma posição em bytes diferentes, um número par de vezes, este erro deixa de ser contabilizado, e passa despercebido pela camada de ligação.

Conclusões

Este trabalho, realizado no âmbito de Redes de Computadores, foi uma mais valia para a nossa aprendizagem, tanto sobre comunicações entre dois dispositivos, como também na programação deste tipo de software. O conhecimento adquirido certamente tem muito valor não só para esta UC como para outras que eventualmente teremos no nosso percurso académico.

Foi muito interessante aprender também sobre correção de erros, máquinas de estados, e estruturas de comunicação entre computadores.