RCOM TP1

Turma 3 // Grupo 3

RCOM 2020/2021

MIEIC FEUP

João de Jesus Costa
up201806560

João Lucas Silva Martins
up201806436

November 8, 2020

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Contents

Chapter 1

Sumário

// TODO

Chapter 2

Introdução

Este relatório incide sobre o projecto desenvolvido para a unidade curricular de RCOM. Neste projecto foi pedido o desenvolvimento de uma aplicação, em linguagem C, que permitisse o envio de ficheiros, entre dois computadores, através de uma *serial port*.

Além disso, esta aplicação devia estar organizada em camadas independentes (discutidas mais à frente) e deve ser resistente a ruído e desconexão durante o envio de informação.

// TODO Dizer o q cada chapter tem

Chapter 3

Arquitetura

O código encontra-se dividido em duas partes principais: a camada de aplicação e a camada de ligação de dados. Por sua vez, estas duas camadas dividem-se na sua componente pública, que é utilizada por um agente externo (**interface**), e a sua componente privada que integra as funções internas da camada.

Chapter 4

Estrutura do código

4.1 Camada de aplicação

A API da camada de aplicação é implementada nos ficheiros *app_layer.h* e *app_layer.c*. Para utilizar esta camada é necessário instanciar um objeto **struct applicationLayer**.

```
struct applicationLayer {
  int fd; /* file descriptor correspondente a porta serie */
  enum applicationStatus status; /* TRANSMITTER or RECEIVER */
  char file_name[256];           /* name of file to transmit (if any) */
  long file_size;                /* size of file to transmit (if any) */
  long chunksize;                /* tansmission chunksize */
};
```

A função **initAppLayer** inicia um objeto do tipo **struct applicationLayer** e a sua correspondente **struct linkLayer**.

```
void initAppLayer(struct applicationLayer *appLayer, int baudrate,
                  long chunksize);
```

A função **llopen** abre uma conexão na dada *serial port*.

```
int llopen(int porta, enum applicationStatus appStatus);
```

A função **llwrite** envia o dado *buffer* através da ligação pré estabelicida.

```
int llwrite(int fd, char *buffer, int length);
```

A função **llread** recebe um pacote da ligação pré-estabelecida para *buffer*.

```
int llread(int fd, char **buffer);
```

A função **llclose** fecha uma ligação anteriormente estabelecida.

```
int llclose(int fd, enum applicationStatus appStatus);
```

A função **sendFile** lê e envia o ficheiro especificado pela camada de aplicação.

```
int sendFile(struct applicationLayer *appLayer);
```

A função **sendFile** recebe o conteúdo de um ficheiro transmitido à camada de aplicação, guardando-o em *res*.

```
int receiveFile(struct applicationLayer *appLayer, unsigned char **res);
```

A função **write_file** cria um ficheiro com o conteúdo de *file_content* e com nome especificado pela camada de aplicação.

```
void write_file(struct applicationLayer *appLayer, unsigned char *file_content);
```

4.2 Camada de ligação

A API da camada de ligação é definida nos ficheiros *data_link.h* e *data_link.c*. Para utilizar esta camada é necessário instanciar um objeto do tipo **struct linkLayer**.

```
struct linkLayer {
  char port[20];                /* Dispositivo /dev/ttySx, x = 0, 1 */
  int baudRate;                 /* Velcidade de transmissao */
  unsigned int sequenceNumber;  /* Numero de sequencia da trama: 0, 1*/
  unsigned int timeout;         /* Valor do temporizador, e.g.: 1 sec */
  unsigned int numTransmissions; /* Numero de retransmissoes em caso de falha */
  vector *frame;
};
```

A função **initLinkLayer** inicializa uma camada de ligação e o seu respetivo **vector**.

```
struct linkLayer initLinkLayer();
```

A função **initConnection** estabelece uma conexão do tipo indicado em *isReceiver* na camada de ligação dada.

```
int initConnection(struct linkLayer *linkLayer, int fd, bool isReceiver);
```

A função **endConnection** fecha uma conexão do tipo indicado em *isReceiver* na camada de ligação dada.

```
int endConnection(struct linkLayer *linkLayer, int fd, bool isReceiver);
```

A função **getFrame** recebe uma trama através de uma conexão pré-estabelecida.

```
int getFrame(struct linkLayer *linkLayer, int fd, unsigned char **packet);
```

A função **getFrame** envia uma trama para uma conexão pré-estabelecida com o tamanho *len*.

```
int sendFrame(struct linkLayer *linkLayer, int fd, unsigned char *packet,
              int len);
```

Chapter 5

Casos de usos principais

A nossa solução para a proposta de trabalho contempla dois casos de uso: o envio de um ficheiro como emissor e a receção do mesmo como recetor. Este modo de funcionamento deve ser referido na chamada do programa através da flag *-s RECEIVER* ou *-s TRANSMITTER*.

Para o programa ser executado como recetor é obrigatório especificar o número da porta série a usar.

Se o programa for executado como emissor, será necessário especificar o caminho para o ficheiro a enviar, em adição aos outros argumentos já enunciados anteriormente. Enquanto emissor, também pode ser útil referir o *chunksize* a usar.

Chapter 6

Protocolos

6.1 Protocolo de ligação lógica

Este protocolo e responsável pela preparação da informação a enviar pela *serial port* e os mecanismos de recuperação de erros/falhas.

Este protocolo transmite informação através do uso de tramas. Existem três tipos de tramas: informação, supervisão, e não numeradas. As tramas de informação carregam os dados a enviar. As tramas de supervisão indicam se um dada de trama de informação foi bem (ou mal) recebida. As tramas não numeradas servem para implementar os mecanismos de inicio e conclusão de uma conexão.

O cabeçalho de cada trama é protegida um BCC. As tramas de informação têm um segundo BCC que protege a informação transmitida. // TODO code snipet do BCC2

// TODO falar bit sequência

De modo a garantir que o byte correspondente à *flag* (byte que indica o inicio e o final de um trama) não ocorre no campo de informação ou no BCC, é realizado o byte *stuffing* (e consequentemente *destuffing*) destas. // TODO code snippet stuffing

Este protocolo faz uso de uma máquina de estados de modo a conseguir recuperar de situações de erro e saber o que fazer com a informação recebida. // TODO snippet state machine e enums dela

6.2 Protocolo de aplicação

Chapter 7

Validação

Chapter 8

Eficiência do protocolo de ligação de dados

Com o objetivo de medir de forma fiável a eficiência do protocolo, foi calculado o tempo de propagação T_{prop} através da fórmula $T_{prop} = T_{absRecebido} - T_{absEnviado}$. Sendo $T_{absRecebido}$ e $T_{absEnviado}$ o tempo absoluto de envio do byte de teste, e o tempo de receção do byte de teste, respetivamente.

Table 8.1: Cálculo do tempo de propagação médio

$T_{absEnviado}(ns)$	$T_{absRecebido}(ns)$	$T_{prop}(ns)$	μT_{prop}
267368335	264536179	2832156	
469990377	467159729	2830648	
672241506	669431417	2810089	2819355
874435168	871625091	2810077	
77204397	74385042	2819355	

Chapter 9

Conclusões