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.

A informação é transmitida em 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.

As tramas começam e terminam por um byte de *flag* (0x7E) e são identificadas por um bit de sequência que oscila entre 0 e 1. Este bit é útil para que o protocolo consiga garantir que nenhuma trama foi perdida ou repetida.

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.

```
int stuffByte(unsigned char byte, unsigned char res[]) {
  int bytes_returned = 1;
  if (byte == ESC) {
    res[0] = ESC;
    res[1] = ESC ^ STUFF_BYTE;
    bytes_returned = 2;
  } else if (byte == FLAG) {
    res[0] = ESC;
    res[1] = FLAG ^ STUFF_BYTE;
    bytes_returned = 2;
  } else
    res[0] = byte;
  return bytes_returned;
}

int stuffString(unsigned char *str, unsigned char *res, int size) {
  int j = 0;
  for (int i = 0; i < size; ++i) {
    unsigned char stuffedBytes[2];
    int n = stuffByte(str[i], stuffedBytes);
    res[j++] = stuffedBytes[0];
    if (n > 1)
      res[j++] = stuffedBytes[1];
  }
  return j;
}

unsigned char destuffByte(unsigned char byte) { return byte ^ STUFF_BYTE; }
```

O cabeçalho de cada trama é protegido por um BCC. As tramas de informação têm um segundo BCC que protege a informação transmitida.

```
unsigned char calcBCC2Field(unsigned char *buf, int size) {
  unsigned char ret = 0;
```

```
  for (int i = 0; i < size; ++i)
    ret ^= buf[i];
  return ret;
}

bool checkBCC2Field(unsigned char *buf, int size) {
  return calcBCC2Field(buf, size) == buf[size];
}
```

Este protocolo faz uso de uma máquina de estados de modo a conseguir interpretar os bytes recebidos. Esta interpretação facilita o *byte destuffing*, a recuperação de erros e a identificação do final de tramas.

```
/* STATE MACHINE */
typedef enum {
  FLAG_RCV = 0,
  A_RCV = 1,
  CI_RCV = 2,
  CS_RCV = 3,
  BCC_RCV = 4,
  OTHER_RCV = 5
} transitions;

typedef enum {
  START_ST = 0,
  FLAG_ST = 1,
  A_ST = 2,
  CS_ST = 3,
  BCC_ST = 4,
  CI_ST = 5,
  D_ST = 6,
  STOP_ST = 7
} state;

static state state_machine[][6] = {
/*  F_RCV     A_RCV     CI_RCV    CS_RCV    BCC_RCV   OTHER_RCV */
  { FLAG_ST,  START_ST, START_ST, START_ST, START_ST, START_ST},  // START_ST
  { FLAG_ST,  A_ST,     START_ST, START_ST, START_ST, START_ST},  // FLAG_ST
  { FLAG_ST,  START_ST, CI_ST,    CS_ST,    START_ST, START_ST},  // A_ST
  { FLAG_ST,  START_ST, START_ST, START_ST, BCC_ST,   START_ST},  // CS_ST
  { STOP_ST,  START_ST, START_ST, START_ST, START_ST, START_ST},  // BCC_ST
  { FLAG_ST,  START_ST, START_ST, START_ST, D_ST,     START_ST},  // CI_ST
  { STOP_ST,  D_ST,     D_ST,     D_ST,     D_ST,     D_ST    },  // D_ST
  { STOP_ST,  STOP_ST,  STOP_ST,  STOP_ST,  STOP_ST,  STOP_ST }   // STOP_ST
};
```

6.2 Protocolo de aplicação

Este protocolo é responsável por abrir a porta série, ler/escrever o ficheiro transferido. Para isso, inicia a camada de ligação de dados, faz a segmentação do ficheiro a transmitir, interpreta a informação recebida e escreve a informação para o ficheiro resultante.

Os pacotes destes protocolo possuem um cabeçalho que indica o tipo de mensagem que contém: controlo (tamanho ficheiro e nome do ficheiro) ou bloco de informação. No caso dos pacotes de informação, o cabeçalho contém um byte de sequência que indica a ordem dos pacotes, começando em zero. A cada pacote recebido este número deve aumentar por um e voltar a zero após o número 255. Se esta não for comprida, é evidente que ocorreu um erro grave e que a execução deve ser terminada.

No lado do emissor, o ficheiro é lido e, em seguida, enviado em blocos de tamanho *chunksize* bytes (valor configurável). No lado do recetor, o ficheiro é lido em blocos de tamanho *chunksize* definido pelo emissor. Quando a transmissão termina, a informação lida é escrita no ficheiro.

O primeiro e último pacotes transmitidos pelo protocolo são pacotes de controlo. Estes pacotes contém informação relativa ao nome do ficheiro e o seu tamanho e um byte identificador. Estes dois pacotes devem ser

idênticos, à excepção do byte identificador (um sinalizará o inicio e o outro o fim da transmissão).

Chapter 7

Validação

De modo a validar o correto funcionamento do programa, enviamos diversos ficheiros com tamanhos e formatos distintos, fazendo variar o valor do *baudrate* e do *chunksize*.

// TODO mostrar tempos

Chapter 8

Eficiência do protocolo de ligação de dados

Chapter 9

Conclusões