• U  C •

Informática Médica

Trabalho 1

João Tiago Fernandes nº2011162899
Noé Godinho nº 2011159459

29 de Outubro de 2015

Índice

1 Introdução

O objetivo deste trabalho é implementar um programa que comunique com equipamentos de monitorização, nomeadamente *V24* e *V26* da *Philips Medical Systems*.

O programa foi feito na linguagem *Java* e era necessário implementar as seguintes mensagens:

- *CONNECT_REQ* e *RSP*
- *DISCONNECT_REQ* e *RSP*
- *PAR_LIST_REQ* e *RSP*
- *SINGLE_TUNE_REQ* e *RSP*

2 Estrutura do Programa

Nesta secção pretende-se descrever sucintamente a estrutura do programa e as soluções encontradas para tanto enviar como receber informação.

Em todos os *requests* que foram efetuados, usou-se um procedimento muito semelhante, que envolve a declaração de algumas variáveis (visto que todos eles têm campos semelhantes no cabeçalho - *DST_ID*, *SRC_ID* e *LENGTH* - estes campos são depois alterados consoante as necessidades de cada comando).

No que toca à leitura (*responses*), recorreu-se à classe *Semaphore.java* para se poder controlar esta situação, ou seja, quando uma *request* acaba de ser feito, um Thread de leitura é notificada e começa a tentar ler do buffer (de 100 em 100 ms). Assim que for lida uma mensagem, esta é processada dependendo do seu tamanho (se é uma resposta do *PAR_LIST* ou do *SINGLE_TUNE*).

2.1 Organização dos métodos e classes implementadas

Neste trabalho, foram alterados dois ficheiros dos fornecidos.

O primeiro a ser alterado foi o **appInterface.java**, em que as únicas alterações feitas foi ativar o botão do *SINGLE_TUNE* e a *textarea* que permite adicionar o *ID* necessário para o *SINGLE_TUNE*. Também foi acrescentado uma verificação que não permita ao utilizador enviar um *ID* vazio.

Depois, foi alterado o **CMSInterface.java**. Este ficheiro contém duas classes e vários métodos por classe:

- **CMSInterface**: Esta classe implementa a escrita e processamento das mensagens a enviar.
  - **changeBytesPosition**: Este método recebe como parâmetro um *array* de 2 *bytes* e retorna-o com os *bytes* invertidos.
  - **setVar**: Este método recebe como parâmetro as das classes **ComInterface** e **appInterface** e atribui-as nas variáveis globais da classe correspondentes. Estas variáveis são usadas para enviar as mensagens ou escrever na interface.
  - **createArrayByteToSend**: Este método recebe um inteiro, sendo o tamanho do *array* final e um *short*, sendo o tipo de *request* a colocar na mensagem e junta a estrutura base nesse mesmo *array* e retorna-o.
  - **createMiniByteArray**: Este método cria um *array* de *bytes* e insere o *short* recebido.
  - **addToArray**: Este método insere um *array* de 2 *bytes*, a partir da posição pretendida, num *array* recebido.
  - **sendMessageToCom**: Este método escreve a mensagem para o servidor, inserindo o *byte* inicial (0x1b) e o *array* recebido e faz *take* do semáforo, de maneira a ser possível ler a resposta do servidor.
  - **connect**: Este método é chamado quando o botão de *connect* é clicado na interface e envia a mensagem *CONNECT_REQ*.
  - **disconnect**: Este método é chamado quando o botão de *disconnect* é clicado na interface e envia a mensagem *DISCONNECT_REQ*.
  - **getParList**: Este método é chamado quando o botão de *Par List* é clicado na interface e envia a mensagem *PAR_LIST_REQ*.
  - **singleTuneRequest**: Este método é chamado quando o botão de *Single Tune* é clicado na interface e é inserido o *ID* e envia a mensagem *SINGLE_TUNE_REQ*.
- **ReadStuff**: Esta classe estende uma *thread* e implementa a leitura, processamento e escrita de mensagens recebidas na interface.
  - **run**: Este método faz *override* ao método *run* da classe *Thread* e começa a ler do servidor quando o semáforo é libertado. Uma lida a mensagem, vai processá-la e escrevê-la no ecrã, de maneira a identificar qual é o campo. Por último, bloqueia o semáforo, de maneira a esperar pela próxima vez que seja necessária a leitura.

- **responseASCIIConversion**: Este método tem como objetivo identificar o campo que está a ser processado no momento, de maneira a ser possível identificar. Recebido uma posição, vai retornar o nome da mesma, conforme o tipo de mensagem recebida a ser processada.
- **processParListMessage**: Este método processa exclusivamente o *PAR_LIST_RSP*. Como esta *response* é composta por várias mensagens e, por sua vez, em cada leitura, é necessário processar a mesma de maneira diferente. Quando o atributo *isParList* está a *true*, será chamada esta função, que tem as variáveis com a informação do número de mensagens totais, do número de mensagem atual, do tamanho total da mensagem actual e do número de *bytes* já lidos da mensagem. No entanto, a causa de um *bug* que não foi descoberto, a última mensagem do *PAR_LIST_RSP* só é impressa depois de clicar no botão *Par List* da interface.
- **processResponse**: Este método processa todas as *responses* (à excepção da *PAR_LIST_RSP*). Tal como o método anterior, inverte os *bytes* 2 a 2, já que ao receber estão trocados, mas só trata uma *response*, que é a que necessita.