Universidade Federal do Ceará – Campus de Quixadá
Disciplina: QXD0043 - Sistemas Distribuídos
Cursos: SI, RC, ES, CC, EC e DD
Professor: Rafael Braga

Trabalho 1 – Comunicação entre processos (Capítulo 4)

Sockets e Streams

1. Cada estudante deve definir um serviço remoto, sugestões. Crie pelo menos classes do tipo POJO de acordo com o serviço escolhido que representem as informações que existem do serviço escolhido e 2 classes de modelo que implementam serviços.

2. Escolha uma das classes POJO do serviço remoto selecionado e transforme essa classe uma subclasse de OutputStream, *por exemplo*, *PojoEscolhido*OutputStream que envia os dados de um conjunto (array) de qualquer outro POJO criado anteriormente seguindo a seguinte regra:

a) O construtor da subclasse deve receber como parâmetros: (i) um array de objetos que representam os dados a serem transmitidos; (ii) o número de Objetos que terão dados enviados pelo stream; (iii) para cada Objeto, deve ser enviada o **número de bytes utilizados para gravar**, pelo menos, 3 atributos e (iv) um OutputStream de destino, para onde as informações do conjunto de objetos devem ser enviadas.

b) Teste sua implementação considerando como OutputStream de destino:

i. a saída padrão (System.out);
ii. um arquivo (FileOutputStream);
iii. um servidor remoto (TCP).

3. Crie uma subclasse para InputStream chamada *PojoEscolhido*InputStream que lê os dados gerados pelo stream do exercício anterior.

a) O construtor da subclasse deve receber como parâmetro um InputStream de origem, de onde as sequências de bytes serão lidas.

b) Teste sua implementação utilizando como origem a entrada padrão (System.in)

c) Teste sua implementação utilizando como origem um arquivo (FileInputStream).

d) Teste sua implementação utilizando como destino um servidor remoto (TCP).

Serialização

4. Implemente um serviço remoto através da comunicação cliente-servidor. A comunicação entre cliente e servidor deve ser implementada via *sockets* (TCP ou UDP) que trocam fluxos de bytes.

As estruturas de dados devem ser 'serializadas' para serem enviadas em mensagens, ou seja, deve ser feito o empacotamento e desempacotamento das mensagens no lado cliente e no lado servidor, ou seja:

- O cliente deve empacotar a mensagem de **request** antes de enviar para o servidor;
- O cliente deve desempacotar a mensagem de **reply** enviada pelo servidor**;**
- O servidor deve desempacotar a mensagem de **requisição** do cliente;
- O servidor deve empacotar a mensagem de **reply** e enviar para o cliente;

Representação externa de dados

5. Implemente uma aplicação distribuída que suporte um sistema de votações. O envio de votos têm um prazo máximo, finalizado esse tempo o servidor não aceita mais votações e calcula o total de votos, respectivas percentagens e candidato ganhador. O eleitor começa a votação através de um login. Em resposta, o servidor envia-lhe uma lista de candidatos que estão em votação, o que permitirá ao eleitor votar em um determinado candidato. Além dos eleitores, existem também os administradores do sistema que têm a capacidade de introduzir e remover candidatos para votação e que poderão enviar notas informativas para os eleitores.

O login, envio da lista de candidatos e de votos deverá ser implementada usando comunicação *unicast* com API de *sockets* TCP em Java, enquanto que toda a comunicação *multicast* será feita utilizando *sockets* UDP. O multicast será utilizado exclusivamente para as notas informativas enviada pelos administradores do sistema. O servidor deverá ser *multi-threaded*.

Para a representação externa de dados nas chamadas remotas (métodos, argumentos e resultados), sugere-se que seja implementada através de *protocol buffers*. Contudo, versões em **XML** ou **JSON** também são aceitas.

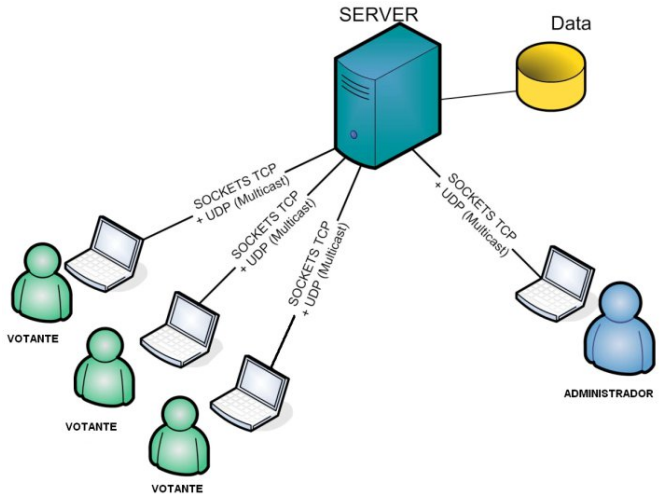

Figura 1: Arquitetura da aplicação, utilizando Java Sockets

Bom trabalho!