1. Introdução

Esta aula TP será sobre a API de Java para comunicação UDP Multicast e abordará o emprego de *threads*

O link para o guião está disponível no email que vos enviei (a leitura do documento presente não dispensa a leitura do referido guião).

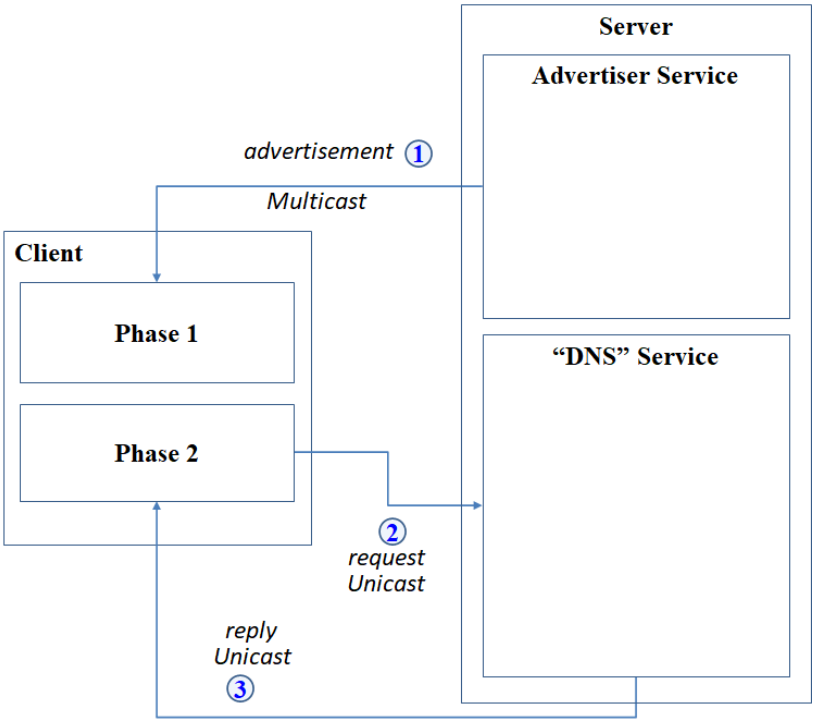

Figure 1 - Vista Geral da Arquitectura a Implementar

Os detalhes básicos da aplicação a desenvolver são os seguintes:

- expande o trabalho da primeira aula (ver Figure 1):
  - mantém o serviço de resolução de nomes DNS – o servidor mantém um serviço de registo e resolução de nomes DNS num determinado endereço IP e numa determinada porta;
  - adiciona uma nova componente de comunicação por *multicast* sobre UDP – o servidor faz *multicast* periódico de uma mensagem onde publicita o endereço e a porta onde disponibiliza o serviço de registo de matrículas (ou seja, o serviço anterior);
- fase 1 – o cliente coloca-se à escuta no endereço de multicast e recebe a mensagem difundida pelo servidor de onde extrai o endereço e porta onde pode aceder ao serviço de registo;
- fase 2:
  - o cliente envia uma mensagem com um pedido ao servidor, na forma de um datagrama UDP;
  - servidor retorna a resposta também como um datagrama UDP.

Os procedimentos e interacções (cliente-servidor) a serem desenvolvidos na fase 1 são apresentados na Figure 2.

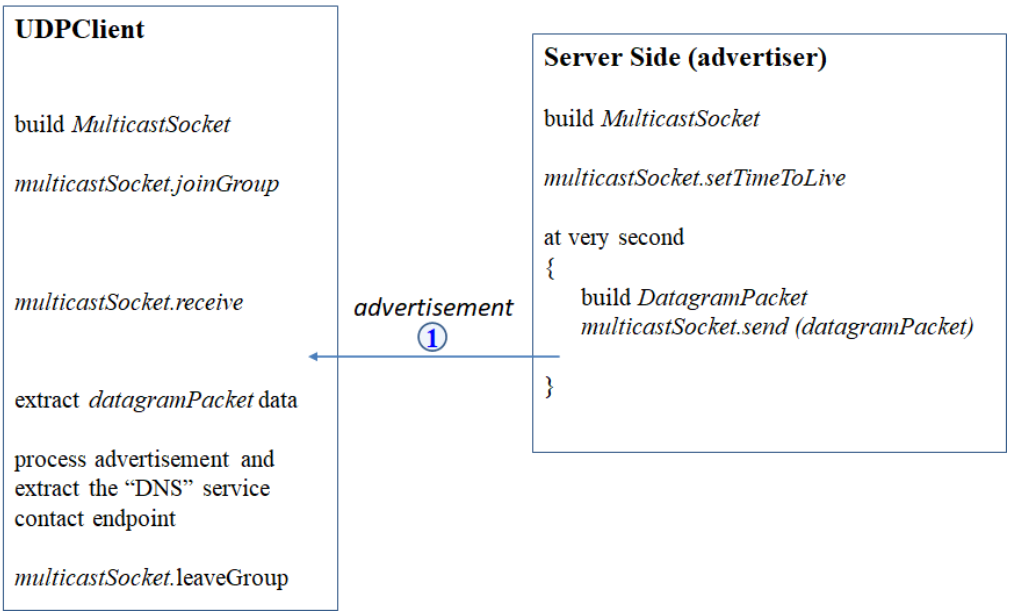

Figure 2 – Fase 1

Na fase 2 são desenvolvidos os mesmos procedimentos e interações descritos para o lab 1, e reapresentados na Figure 3.

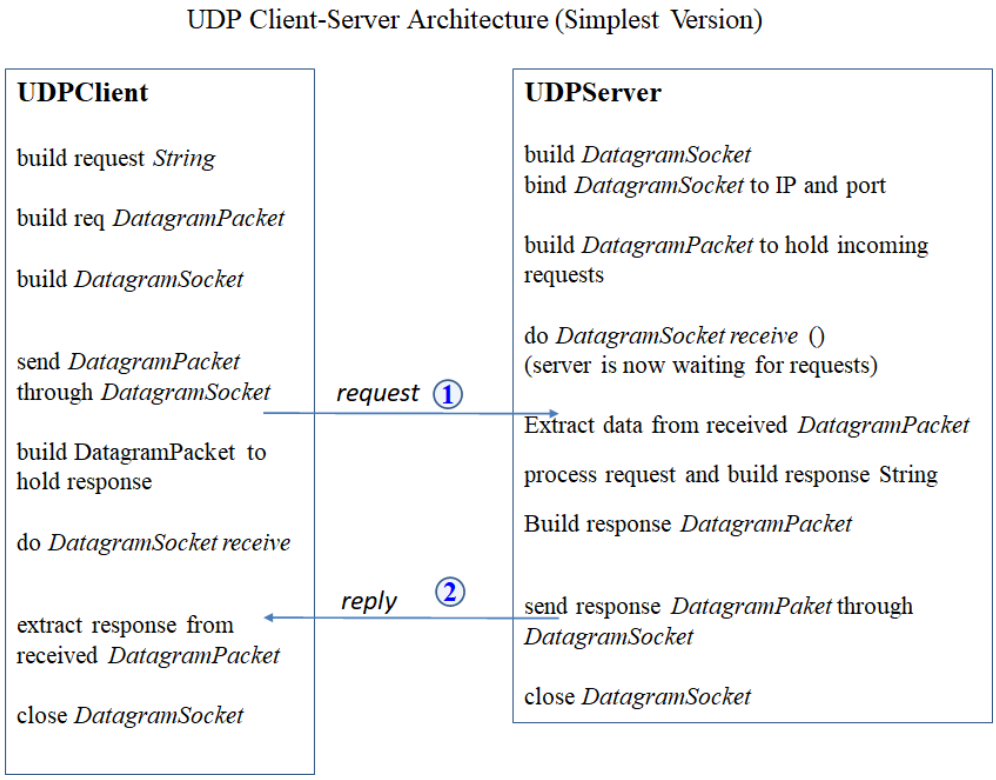

Figure 3 – Fase 2

2. API Java para UDP e outras Classes Relevantes

Na API Java para comunicação sobre UDP as classes mas relevantes são:

- ***java.net.DatagramPacket*** – os datagramas UDP a trocar entre cliente e servidor deverão ser criados empregando instâncias da classe ***DatagramPacket***;

- ***java.net.DatagramSocket*** – os pontos de envio e receção de mensagens unicast, dos dois lados, deverão sem implementados com instâncias da classe ***DatagramSocket***;
- ***java.net.InetAddress*** e as suas (subclasses ***Inet4Address*** e ***Inet6Address***) – esta classe é importante para permitir o uso das duas anteriores;
- ***java.net.MulticastSocket*** – sockets para envio e escuta de datagramas UDP enviados por multicast.

Em relação à classe ***java.net.MulticastSocket***, queria salientar o seguinte:
1) Para receber mensagens multicast, há que usar ***java.net.MulticastSocket*** que deverá ser associado ao grupo multicast.
2) Para enviar mensagens multicast, poderiam usar alternativamente a classe ***java.net.DatagramSocket***, mas esta não permite limitar o ***TimeToLive*** do datagrama (ou seja, o numero de hops, ou dispositivos, através do qual será difundida). Assim o desejável é que usem a classe ***java.net.MulticastSocket*** pois essa permite especificar essa limitação através do método ***setTimeToLive(…)*** *(*deve fazer-se ***setTimeToLive(1))***.

Outras classes relevantes:
- ***java.lang.Thread –***
- ***java.util.Timer*** e ***java.util.TimerTask***
- ***java.util.concurrent.ScheduledExecutorService***

3. Detalhes da Implementação

Podem implementar este lab de uma forma *single-threaded* ou *multi-threaded*.

A primeira forma (que é a mais simples mas menos realista) para implementar isto (com um só *thread*) é usando o método ***setSoTimeout()*** das classes ***Socket***. A ideia é fazer o multicast e ficar à espera dum pedido via unicast, após ter invocado ***setSoTimeout()*** sobre o socket correspondente.

Aquando do "timeout" o servidor faz novo multicast. Caso receba um pedido, deverá responder-lhe após o que faria um novo multicast.

Tem o pequeno problema de no caso de haver pedidos a processar o intervalo entre mensagens multicast não ser de 1 s, conforme pedido.

Garantir o anúncio do serviço com um período muito próximo de 1 s, implica alguma complexidade. Uma solução aproximada é anunciar o serviço após o processamento de cada pedido, se o serviço tiver sido anunciado há mais de 1 segundo. Esta solução obriga à leitura da hora (usando p.ex. a classe ***java.util.Calendar***), podendo neste caso o intervalo entre anúncios consecutivos variar bastante.

Outra forma, a mais aconselhada, é usar 2 threads:
- um que espera por pedidos do serviço
- outro que faz o multicast periódico da mensagem anunciando o serviço.

Há várias implementações possíveis, por ordem crescente de sofisticação:
1) usando a interface básica (classe ***java.lang.Thread***) e o método ***sleep()***
2) usando as classes ***java.util.Timer*** e ***java.util.TimerTask***. Estas classes fornecem uma interface muito simples.
3) usando a classe ***java.util.concurrent.ScheduledExecutorService***

O enunciado tem "links" úteis para as implementações 2) e 3). A minha sugestão é que se centrem na solução 3, que é mais genérica.

A minha sugestão é que implementem primeiro os protocolos e provisões de comunicação, mesmo que o serviço não esteja correto (respostas pré-definidas) ou tenha sérias limitações, (suportando o registo duma única matrícula).

Anexo

A multicast address is a logical identifier for a group of hosts in a computer network that are available to process datagrams or frames intended to be multicast for a designated network service.

IP multicast address range	Description	Routable
224.0.0.0 to 224.0.0.255	Local subnetwork[1]	No
224.0.1.0 to 224.0.1.255	Internetwork control	Yes
224.0.2.0 to 224.0.255.255	AD-HOC block 1[2]	Yes
224.3.0.0 to 224.4.255.255	AD-HOC block 2[3]	Yes
232.0.0.0 to 232.255.255.255	Source-specific multicast[1]	Yes
233.0.0.0 to 233.251.255.255	GLOP addressing[4]	Yes
233.252.0.0 to 233.255.255.255	AD-HOC block 3[5]	Yes
234.0.0.0 to 234.255.255.255	Unicast-prefix-based	Yes
239.0.0.0 to 239.255.255.255	Administratively scoped[1]	Yes