Sistemas Distribuídos

Prof. Alexandre Sztajnberg 2017/2

Projeto – Programação usando sockets e middleware para objetos distribuídos (RMI, CORBA, etc.) ou RPC

O objetivo deste projeto lista é desenvolver um pequeno sistema mensagens curtas para um **grupo** de usuários utilizando *sockets* e algum suporte de middleware (RPC, RMI ou CORBA). A Figura 1 apresente uma ideia abstrata do sistema.

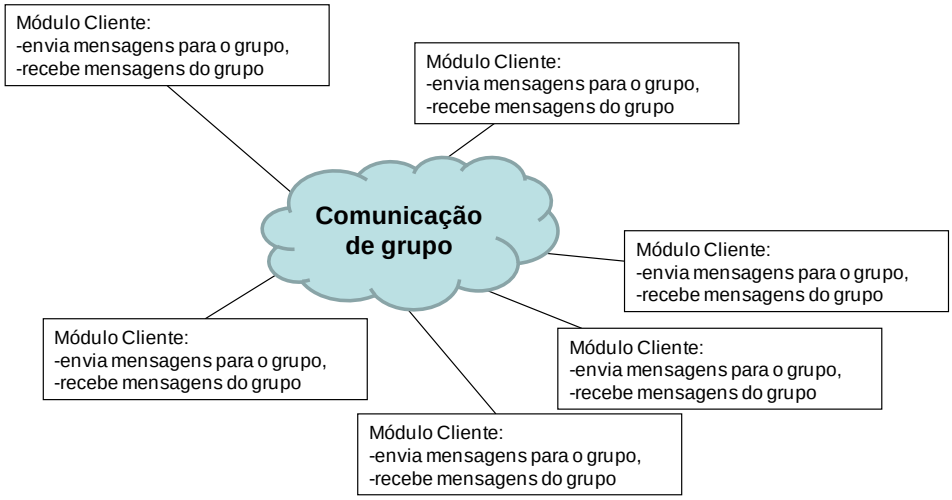

Figura 1. Arquiteutra do sistema de chat em grupo

O projeto será composto de algumas etapas, que serão avaliadas separadamente e também em conjunto.

O sistema/programa a ser implementado pode ser desenvolvido em C, C++, Java ou Python e deve executar no Linux, mais especificamente nas nossas máquinas virtuais 152.92.236.11 e 236.16.

A documentação do sistema deve ser disponibilizada na conta da 236.11 através de uma página Web simples, mas organizada.

O sistema terá duas camadas: uma de comunicação de grupo e outra para o módulo cliente. O módulo cliente deve usar RMI.

O suporte à comunicação de grupo terá que usar RMI para interagir com os módulos clientes (receber mensagens vindas de cada um, destinadas ao grupo + enviar para cada cliente mensagens vindas de algum membro do grupo). Para efetivar a comunicação de grupo deverá usar socket. Pode ser usado socket *multicast* (neste caso, a comunicação é via UDP) ou socket TCP (neste caso, deverá ser formada uma malha de canais *multicast*).

A opção é ilustrada na Figura 2:

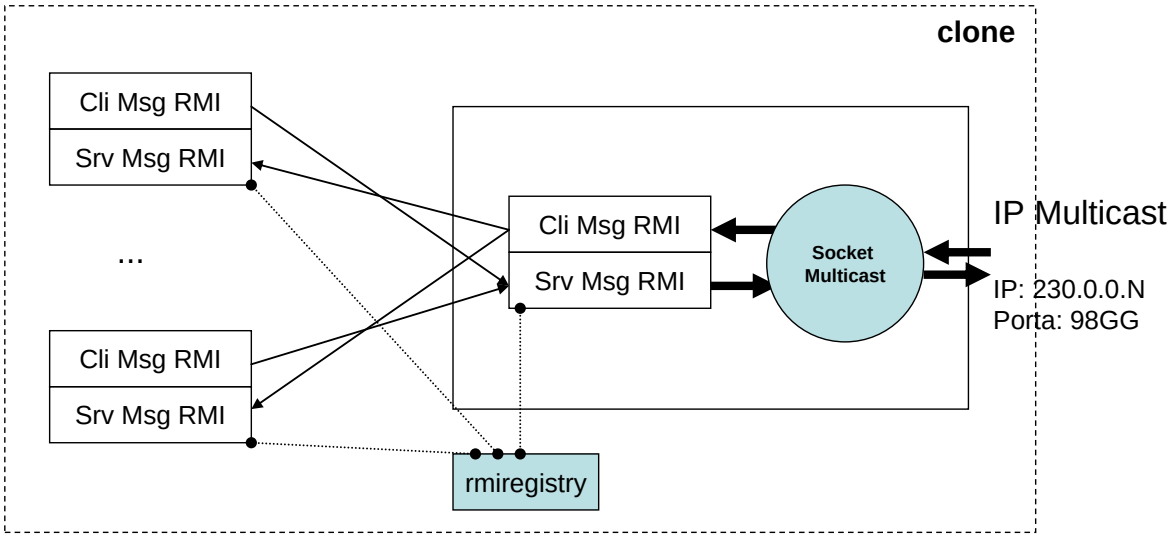

Figura 2. Detalhe da arquitetura com multicast

A opção com TCP, em duas possibilidades de arranjo de conexões é apresentada na Figura 3.

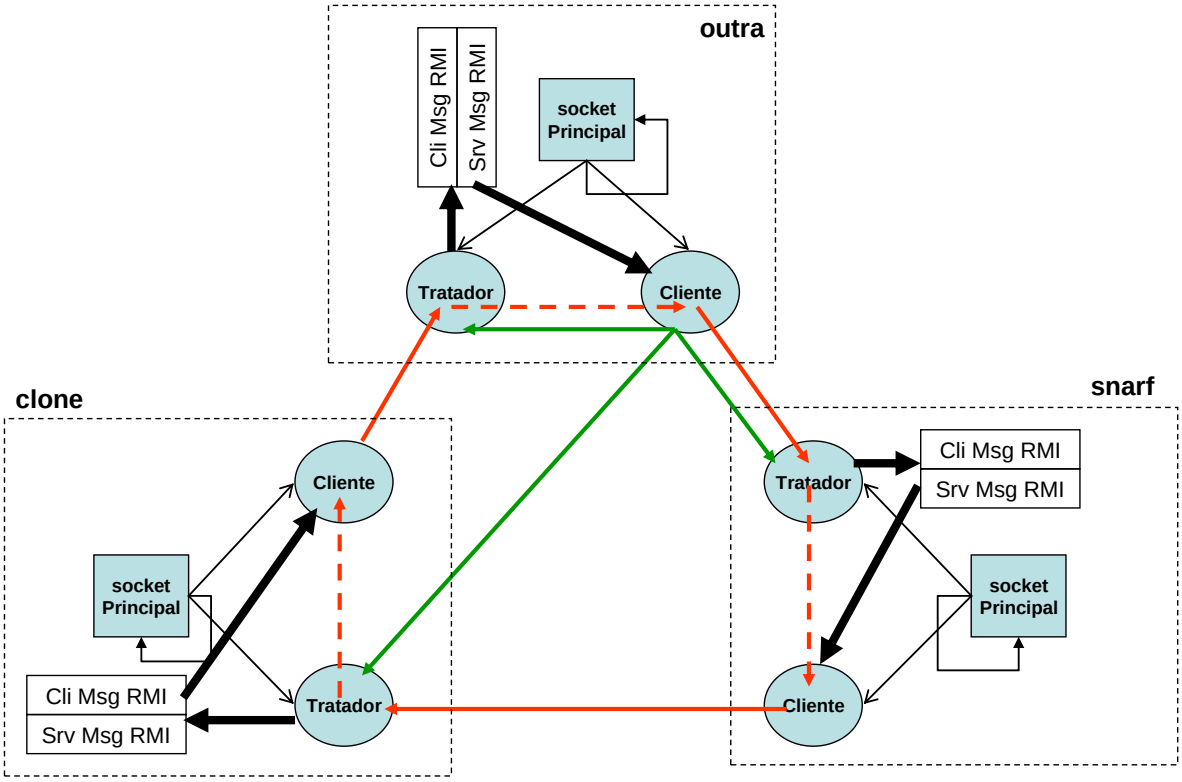

Figura 3. Detalhe da arquitetura com TCP, arranjo circular (vermelho) ou 1:N (verde)

Cada nó deverá ter (cada aluno fará o seu):

- um servidor de comunicação usando as opções da Figura 2 ou 3. Cada servidor terá uma parte RMI e uma parte socket

- um ou mais clientes que se comunicarão por RMI (ou RPC, etc.) com o servidor de comunicação para:
    - enviar mensagens de texto
    - receber mensagens de texto vindos de outros clientes
- tanto dos clientes, como a parte RMI do servidor devem se registrar no *rmiregistry* local (ou servidor de nomes apropriado)

Cada um destes elementos deve ser desenvolvido, implementado e testado. Observe que os módulos podem ser implantados em máquinas distribuídas. Portanto, NÃO deixe referências *hardcoded* na versão final. Use arquivos de configuração ou parâmetros de linha de comando.

Faça scripts de inicialização para colocar o sistema de comunicação (um por nó) e para subir um cliente.

O sistema vai oferecer um serviço de "chat" em grupo. Você vai ter que pensar nas várias interfaces, formatos de mensagens, tratamento de erros e exceções. Principalmente, num sistema de nomes/referência para os clientes, e servidores (RMI e socket).

Um arquivo de texto, tipo README deve descrever as decisões de projeto, entre elas as relacionadas ao endereçamento, registro, portas, obtenção destas informações, etc. O conteúdo deste arquivo também deve estar na página Web do grupo.

Características:

1. Servidor de comunicação

versão multicast. Usa socket UDP, e o número IP classe D. Vantagem, cada sistema de suporte só precisa transmitir um único datagrama para o sistema de suporte do grupo (para ele mesmo e para os outros nós)

versão TCP circular. Usa socket TCP. Estabelecer um "circulo" de conexões e cada mensagem vai circular entre todos os sistemas de comunicação até voltar para a origem. Ao receber a mensagem cada sistema "notifica" seus clientes locais da mensagem que acabou de chegar.

versão TCP 1:N. Nesta versão, cada sistema vai ter que estabelecer uma conexão com cada outro sistema (e até "com si mesmo"). Para transmitir uma mensagem de um cliente a mesma deve ser reenviada para todos os canais.

Um arquivo de configuração deve ser lido e deve conter as configurações a serem usadas (número IP, número de porta, quem se conecta com quem –se for o caso).

Observações: para padronizar, use a porta 98ID (onde ID é o seu número de ID – 01, 02, 03, etc. Se você for usar *multicast*, use o número IP 230.0.0.N) onde N é o número do seu ID – 1, 2, 3 [sem o 0 antes]). Para o servidor de registro do RMI (*rmiregistry*), utilize a porta 99ID, evitando conflitos com a execução de outros alunos.

2. Clientes

O cliente deve interagir com o servidor de comunicação. Ao enviar uma mensagem de texto, a data e hora local devem ser também enviadas.

Ao dar “display” em todas as mensagens a data/hora de envio e data/hora de chegada (ou de display) deve ser exibida.

Obs: Provavelmente será necessário utilizar concorrência, ou através de threads, ou através de vários objetos com objetos compartilhados ...

E o tempo para fazer isso tudo?

Você tem um mês para fazer esse trabalho. Basicamente é um trabalho de integração, já que você tem os exemplos. Numa próxima versão do enunciado podemos destacar os itens a serem avaliados. Obviamente, o funcionamento correto e o atendimento aos requisitos é importante.

Bônus

Talvez não seja possível observar o problema executando o projeto nas duas máquinas virtuais e abrindo janelas SSH e enviando mensagens com um teclado apenas. Mas, talvez fosse o caso de ver o problema acontecer se pudéssemos efetivamente distribuir os servidores de comunicação e termos vários clientes RMI espalhados:

1) pode ser que um ou mais usuários recebam mensagens em ordem diferente
2) pode ser que a hora carimbada pelo enviante esteja muito diferente da hora recebida por alguns usuários.

Bônus. Por que isso acontece e como resolver? Implemente.