Tarefa de Programação 01: Implementação de Referências Remotas

(Este é o mesmo exercício enunciado no Slide 20.)

Após ler a Nota 4.5 (págs. 181 e 182 do livro-texto) e *compreender o princípio da passagem de referências remotas como parâmetro em sistemas distribuídos*, estude a Nota 4.8 (págs. 196-198), que descreve, por meio de um exemplo (o código das Figs. 4.21 (a)-(d)), o uso de stubs como forma de implementar referências remotas. O código completo do exemplo pode ser encontrado no companion website do livro, neste link. Trata-se do código encontrado na pasta 04-21 (mais especificamente, as versões sem a palavra "book" no nome do arquivo).

Esta tarefa consiste em executar o exemplo da Fig. 4.21 para reproduzir o cenário mostrado na Fig. 10, ou seja, um cliente que possui a referência para o objeto remoto passa essa referência para outro cliente, que, com isto passa a ter também a capacidade de fazer chamadas para o mesmo objeto remoto. A execução deve ser feita utilizando três máquinas distintas, alocadas na nuvem AWS (use sua conta do AWS Educate).

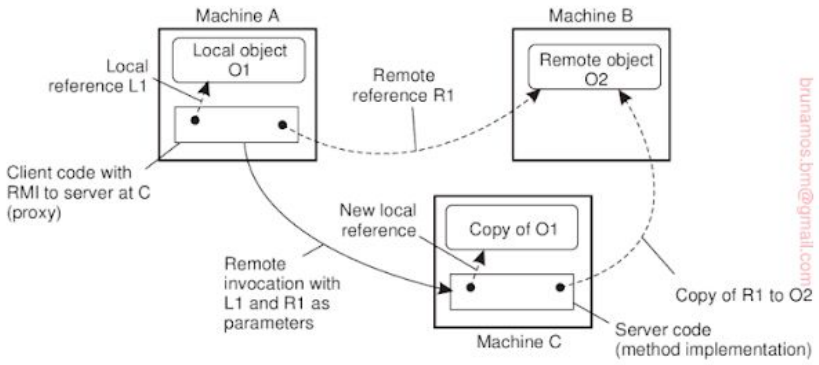


Figure 4.10: Passing an object by reference or by value.

O que entregar:

(1) o código-fonte (empacotado em um arquivo .zip); e

(2) um vídeo (screencast) de sua autoria, com no máximo 4 minutos, que demonstra e explica a execução do exemplo. No vídeo, alterne a exibição da execução com a explicação do código envolvido em cada etapa. Mande apenas o link para o vídeo (utilizando o campo de texto da resposta).

Obs.: guarde bem o código e todas as eventuais alterações que você precisar fazer nele, pois o utilizaremos como base para outra tarefa.

Tarefa de Programação 02: Descoberta automática do servidor de aplicação

Utilize a biblioteca rpyc (cujo uso está exemplificado no slide 22 e foi descrito na aula de 27-10) para implementar o cenário descrito no slide 27 (binding entre cliente e servidor).

1. Nesta tarefa, você deve implementar o servidor de diretório e as interações 2, 3 e 5.
2. O servidor (Server) deve registrar o endereço IP e o número de porta (juntamente com seu nome legível) no servidor de diretório.
3. O cliente usa o nome legível do servidor para descobrir o endereço IP e o número de porta do servidor por meio de uma chamada remota (lookup, via RPC) ao servidor de diretório.
4. Em seguida, o cliente faz uma sequência de chamadas (RPCs) para demonstrar o acesso ao servidor.

O que entregar: Link para o código fonte no GitHub, onde deve haver também um readme com a explicação dos componentes da implementação.

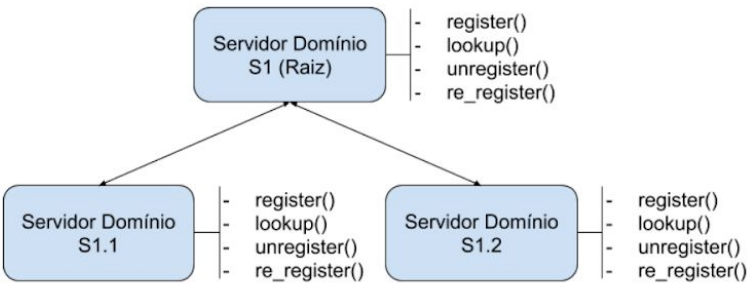
Tarefa de Programação 03: Melhorias no Servidor de Diretório

Partindo do produto final da Tarefa de Programação 02, implemente melhorias no Servidor de Diretório para:
a) Registrar novamente um nome já existente;
b) Evitar o lookup de nomes não existentes;
c) Remover o registro de um nome existente.

Para completar o exercício, acrescente novas funcionalidades ao servidor da aplicação para excluir e consultar elementos da lista.
A forma de entrega é a mesma da Tarefa de Programação 02.

Tarefa de Programação 04: Servidor de Nomes Hierárquico

Utilizando como base o servidor de nomes construído na Tarefa de Programação 3, construa um serviço de nomes hierárquico de dois níveis, como ilustrado na figura abaixo e descrito na sequência.



Este serviço de nomes hierárquico é composto por um servidor de nomes raiz e dois servidores de nomes de segundo nível. Todos os servidores possuem exatamente a mesma interface, com as seguintes operações:

Operation name	***Parameters***	***Return value***	***Description***
register	(name, address)	fully qualified name (or error message)	Register a new name and associate it with the given address (composed by IP address and port number). Returns FQN (fully qualified name, i.e., a complete name in the form context:name) *Note:* in this version of the hierarchical naming service, name registration is always local, i.e., register() must be called exactly on the server where the name will be registered.
lookup	(name)	address (or error message)	Resolves a name. Returns the address (IP and port#) associated with the name. The name parameter may be a simple name, in which case it is resolved locally only, or an FQN, in which case it may be resolved by querying the other naming servers (by following the hierarchy). *Note:* name resolution can be either *local* (where lookup() receives a simple name and is called on the same naming server where the name is registered) or *global* (where lookup() receives an FQN and may be called on a different naming server than the one where the name was registered (in this case, the initial naming server forwards the .
unregister	(name)	Ok message (or error message)	Deletes a previously registered name.
re_register	(name, address)	fully qualified name (or error message)	Updates the address associated with an existing name.

Observações:

- O servidor de nomes raiz não possui nome.
- Cada servidor de nomes de segundo nível possui um nome único, o qual deve ser registrado no servidor de nomes raiz.
- Nomes de servidores de aplicação são sempre registrados localmente nos servidores de nomes de segundo nível, o que significa que o servidor de nomes raiz serve apenas para registrar os nomes dos servidores de segundo nível.
- Servidores de aplicação diferentes podem ser registrados com o mesmo nome em servidores de nomes diferentes. Mas observe que os FQNs (fully qualified name) sempre serão diferentes.
- A resolução de nomes global pode ser feita tanto recursivamente quanto iterativamente; o método escolhido deve ficar claro no relatório da tarefa.
- O servidor de nomes raiz (i.e., seu endereço) deve ser conhecido a priori por todos os clientes. Os servidores de nomes de segundo nível devem ser descobertos a partir do servidor de nomes raiz.

Exemplo:

Para ilustrar, vamos instanciar a figura acima. Imagine que o serviço de nomes hierárquico pertença a uma empresa. Então, os servidores de nomes de segundo nível podem pertencer aos diferentes departamentos da empresa. Assim, S1.1 pode pertencer ao departamento de vendas, enquanto S1.2 pode pertencer ao departamento de recursos humanos. Logo, o nome de S1.1 pode ser "DeptVendas" e o nome de S1.2 pode ser "DeptRH". Estes dois nomes devem ser registrados no servidor de nomes raiz. Agora,

imagine que cada departamento possua um servidor de aplicação que oferece o serviço (hipotético) de calculadora (chamado localmente de "CalcServer"). O servidor de aplicação de cada departamento será registrado localmente (i.e., no respectivo servidor de nomes) com o mesmo nome. Entretanto, eles terão nomes completos (FQNs) distintos, a saber: "DeptVendas:CalcServer" e "DeptRH:CalcServer".

Obs.: Um FQN tem sempre este formato: <NamingContext>:<LocalName>

Para demonstrar o funcionamento do serviço de nomes hierárquico, construa pelo menos três tipos de servidor de aplicação (que podem ser hipotéticos, ou seja, que não precisam ter funcionalidades realísticas). Por exemplo: servidor de calculadora, servidor de previsão do tempo e servidor de hora. Crie instâncias distintas desses servidores de aplicação e as registre com seus respectivos nomes nos dois domínios de segundo nível. Em seguida, demonstre a descoberta de serviços com um cliente que faça uma sequência de consultas locais e consultas globais e, em seguida realiza uma sequência de chamadas aos servidores de aplicação descobertos.

O que entregar:

- Código-fonte no GitHub (URL do projeto)
- Relatório com a descrição da implementação e de um exemplo de uso
- Vídeo de até 3 minutos que demonstre a execução do cliente e dos servidores de nomes.

Dicas AWS

Tutorial: Getting started with Amazon EC2 Linux instances
EC2 - Infraestrutura com serviço - serviço mais fundamental

- console do EC2
    - acesso a nuvem da Amazon para criar recursos, ou seja, para criar máquinas virtuais na nuvem
    - uma instância é um servidor virtual na nuvem
    - as imagens das máquinas disponíveis são configurações de servidores (distribuição de servidor) - não tem interface gráfica...
    - vCPU - quantidade de núcleo da máquina
    - EBS only - configuração da amazon que permite que a memória da máquina se expanda à medida que for necessário.. inicialmente pode começar com 8Gb disponíveis.
        - Esse é um recurso mais barato... pagamento é feito por hora de consumo.
- Grupo de segurança:
    - AWS cria um grupo para cada máquina por default, não permite que haja comunicação entre máquinas
    - o objetivo de um grupo de segurança é realizar uma espécie de verificação dentro da rede que permite que as máquinas criadas possam acessar umas às outras.
    - Só será possível a comunicação entre 2 ou mais máquinas que estão no mesmo grupo de segurança.
    - Em um sistema distribuído é necessário que haja comunicação entre as máquinas.

    - Se a configuração do grupo de segurança permitir a comunicação entre as máquinas, elas irão trocar informações entre si.
        - Políticas de segurança compatíveis
    - posso ter vários grupos de segurança para cada máquina
    - na opção Security Groups
        - Inbound rules - regras de entrada, ou seja, regras que estabelece como uma máquina deve ser comunicação
            - type, protocol, port range, source (grupo de segurança que a máquina que deseja se comunicar deve pertencer) e description
        - Outbound rules - regras de saída, como as informações são enviadas da máquinas
            - normalmente essas conexões são para toda a internet.. não tem muita restrição...
        - no campo source posso aceitar conexão de um grupo específico, ou que qualquer endereço da internet: 0.0.0.0/0 por exemplo...
        - em um grupo de segurança posso ter várias regras...
- Key pair
    - autenticação para logar remotamente em uma máquina, AWS não permite a autenticação por meio de usuário e senha, logo usa um par de chaves para verificar a autenticação.
- Conexão em uma máquina remota
    - acha o arquivo onde esta o par de chaves
    - não deixar a máquina rodando se não for necessário...