RPC - *Remote Procedure Call*

Sistemas Distribuídos e Mobile

Prof. Me. Gustavo Torres Custódio
gustavo.custodio@ulife.com.br

Conteúdo

Remote Procedure Call (RPC)

Chamada de Procedimento Convencional e Chamada Remota

RMI - Java

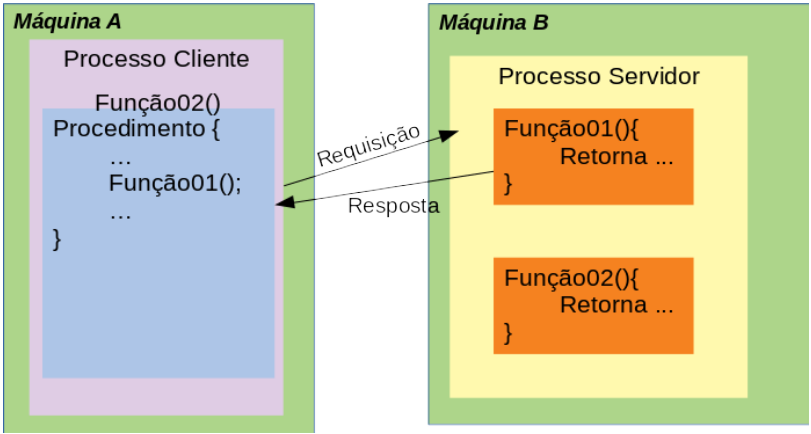
RPC - Remote Procedure Call

Remote Procedure Call (RPC)

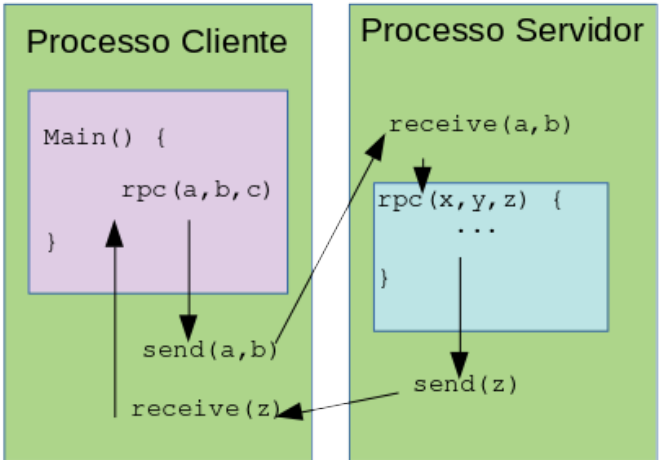
RPC

- *Remote Procedure Call* (RPC) consiste em chamar um **procedimento disponível em outra máquina**.
  - Os parâmetros necessários para executar o procedimento são passados junto com a chamada remota.
  - Retorno das informações para o processo “chamador” (a máquina que pediu para utilizar o método).
  - Transparência - do ponto de vista do usuário, é como se um método comum fosse chamado.

RPC


Máquina A
Processo Cliente
Função02()
Procedimento {
…
Função01();
…
}
Requisição
Resposta
Máquina B
Processo Servidor
Função01(){
Retorna ...
}
Função02(){
Retorna ...
}

RPC


Processo Cliente
Main() {
rpc(a,b,c)
}
send(a,b)
receive(z)
Processo Servidor
receive(a,b)
rpc(x,y,z) {
...
}
send(z)

RPC

- **Processos em máquinas diferentes** chamam diferentes partes de um programa.
- Os processos se comunicam por meio de troca de mensagens.
- A troca de mensagens é feita por meio das primitivas **`send`** e **`receive`**.
  - envio e recebimento.

RPC

- É importante destacar que a chamada por RPC tem natureza síncrona.
  - Ou seja, o cliente em **espera até a resposta ser enviada** pelo servidor.
  - Diferente da comunicação assíncrona, onde outras instruções podem ser executadas enquanto a resposta não é enviada.

Chamada de Procedimento Convencional e Chamada Remota

RPC - Remote Procedure Call

Chamada de Procedimento Convencional

- A passagem de parâmetros é feita **por valor** ou **por referência**.
- Por valor:
  - O parâmetro é copiado para a pilha de execução.
  - O procedimento chamado pode afetar o valor do parâmetro, mas não afeta a variável original.

Chamada de Procedimento Convencional

- Por Referência:
  - O parâmetro é um ponteiro para a variável
  - Se o parâmetro tem seu valor alterado, isso é refletido no processo chamador.

Chamada de Procedimento Remoto

- Para o SO (kernel), o envio e recebimento de mensagens é transparente, ou seja, **ele não sabe que está fazendo uma RPC**.
  - Transparência.
  - Portanto, deve existir um processo específico para o tratamento de RPCs.
  - Isso é feito por meio de *stubs*.
    - Basicamente possuem a função de **converter chamadas remotas para locais**.

Chamada de Procedimento Remoto

- O cliente e o servidor possuem seu próprio stub individual.
  - Stub do Cliente.
  - Stub do Servidor.

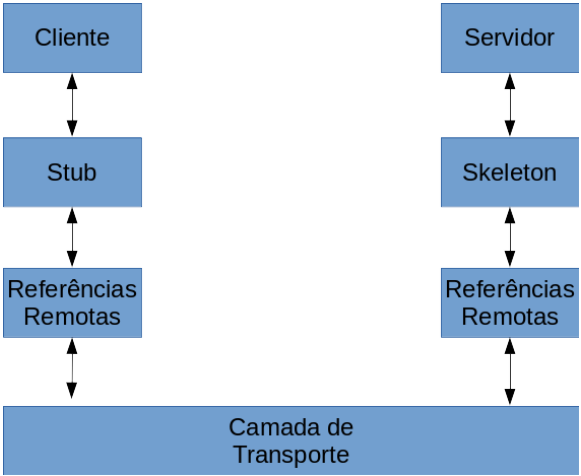
Chamada de Procedimento Remoto

- *Stub do Cliente*:
  - **Empacota os parâmetros em uma mensagem** e a envia para a máquina do servidor.
  - O cliente **bloqueia a si mesmo em *receive*** até que a resposta do servidor chegue (síncrono).

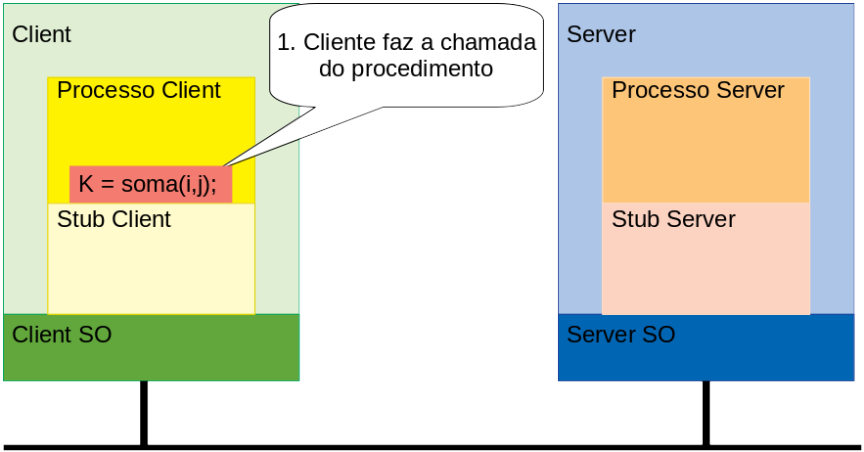
Chamada de Procedimento Remoto

- *Stub do Servidor*:
  - Quando a mensagem chega ao servidor, o SO dele passa a mensagem para o *stub* do servidor.
  - **Desempacota os parâmetros da mensagem do cliente e executa o procedimento** no servidor.
  - **Envia mensagem de volta** ao cliente por meio da instrução primitiva *send*.

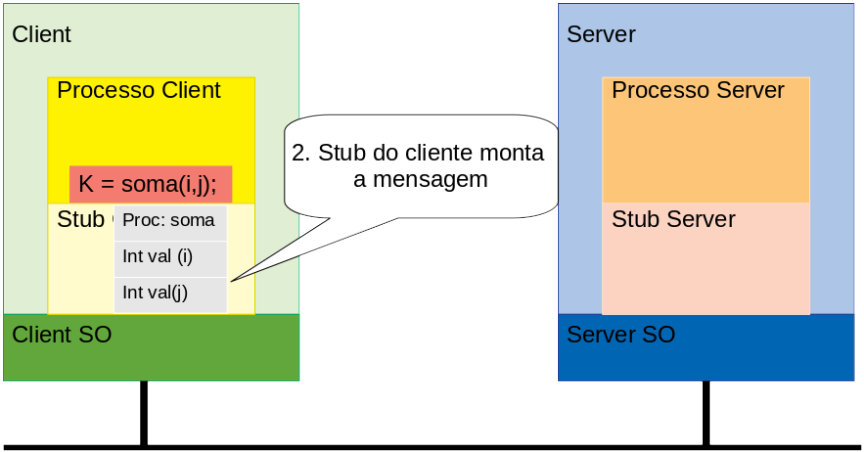
Chamada de Procedimento Remoto



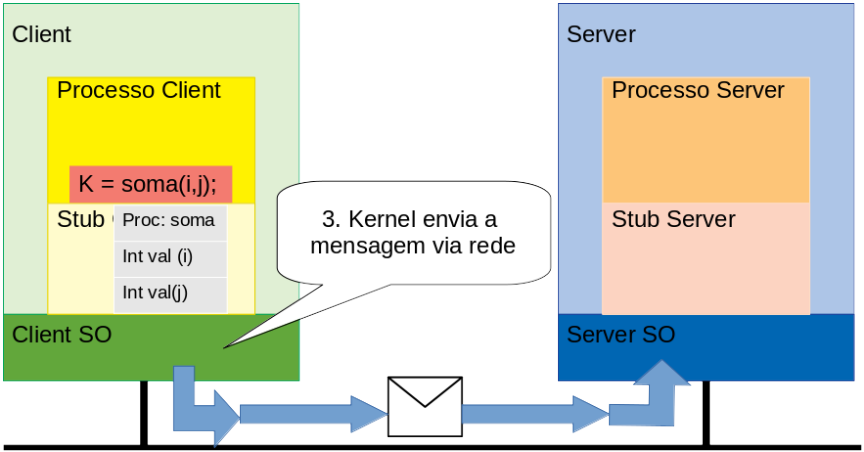
Chamada de Procedimento Remoto



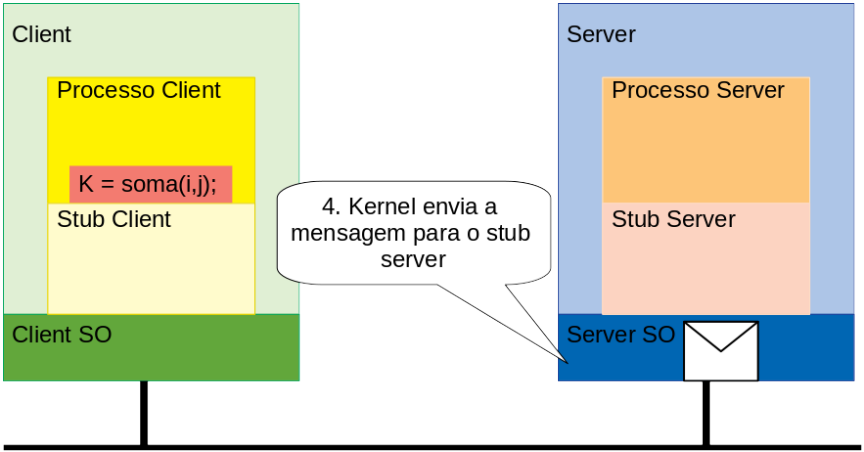
Chamada de Procedimento Remoto



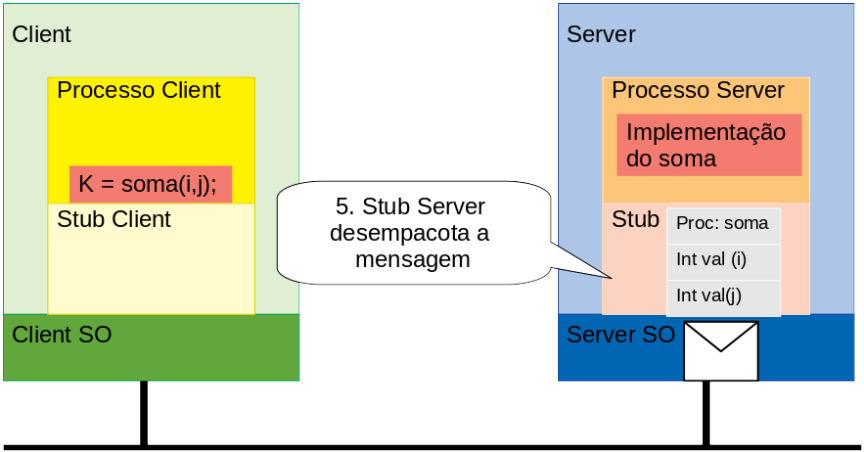
Chamada de Procedimento Remoto



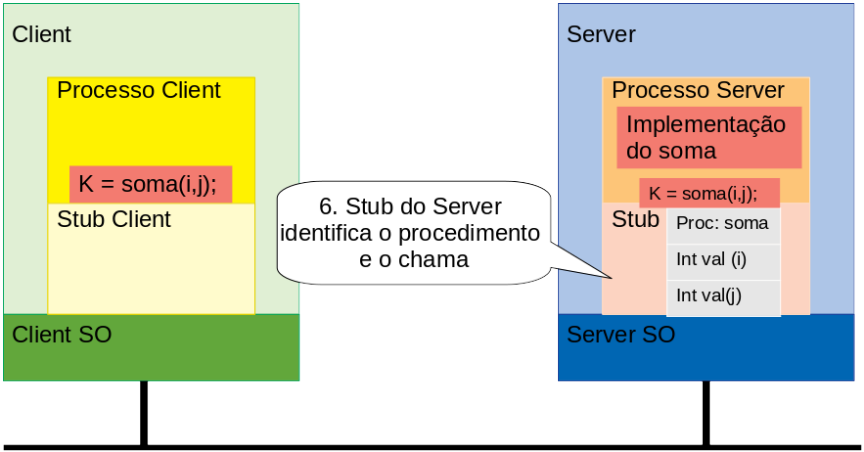
Chamada de Procedimento Remoto



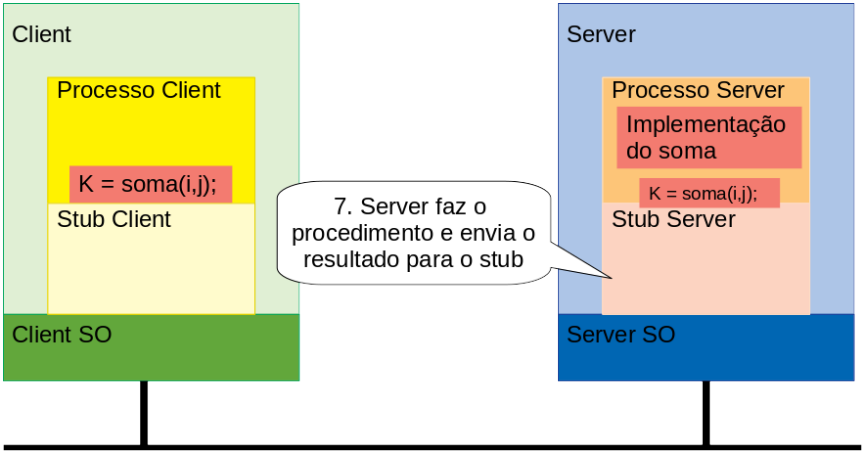
Chamada de Procedimento Remoto



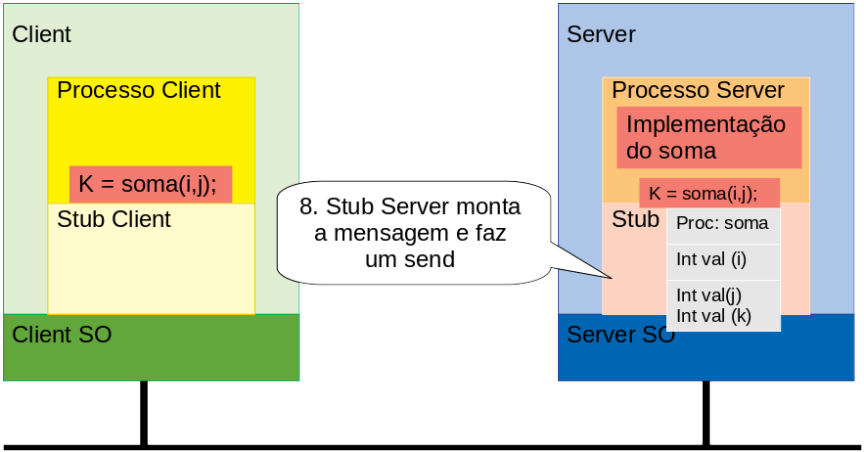
Chamada de Procedimento Remoto



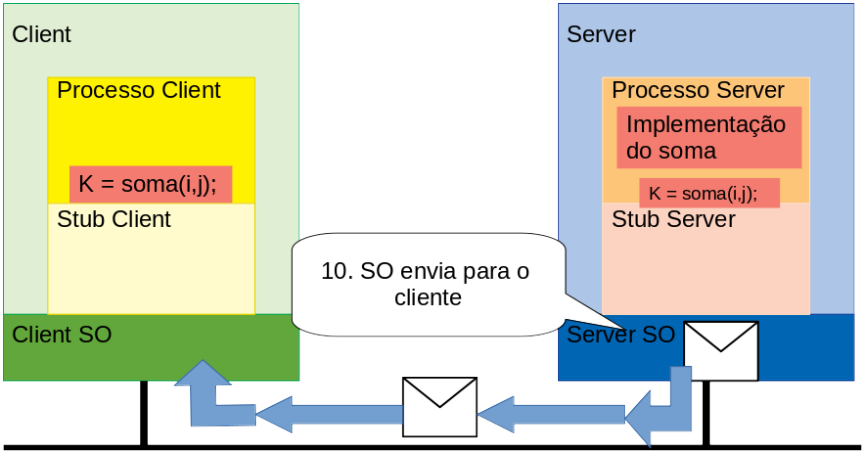
Chamada de Procedimento Remoto



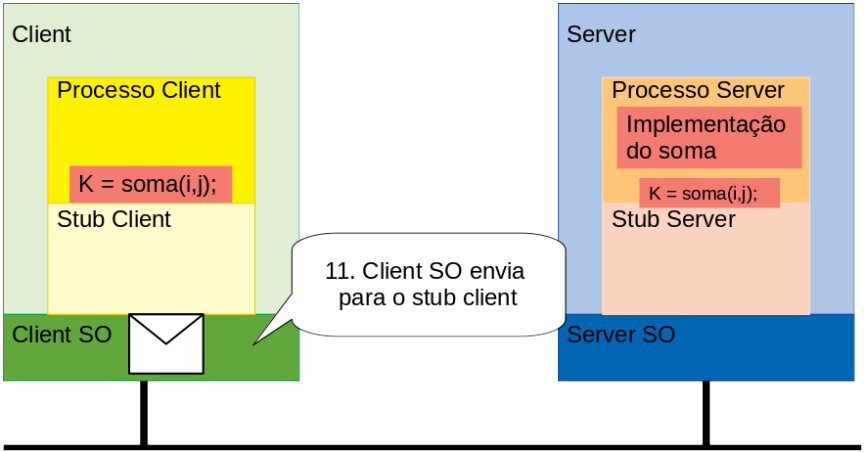
Chamada de Procedimento Remoto



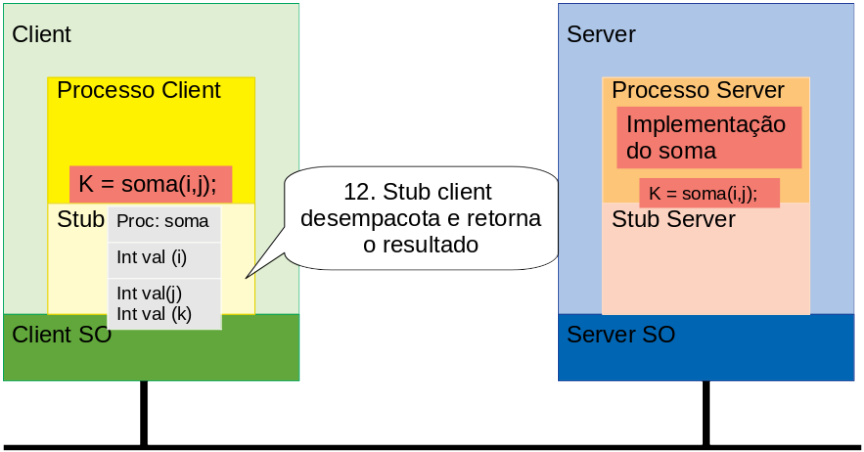
Chamada de Procedimento Remoto



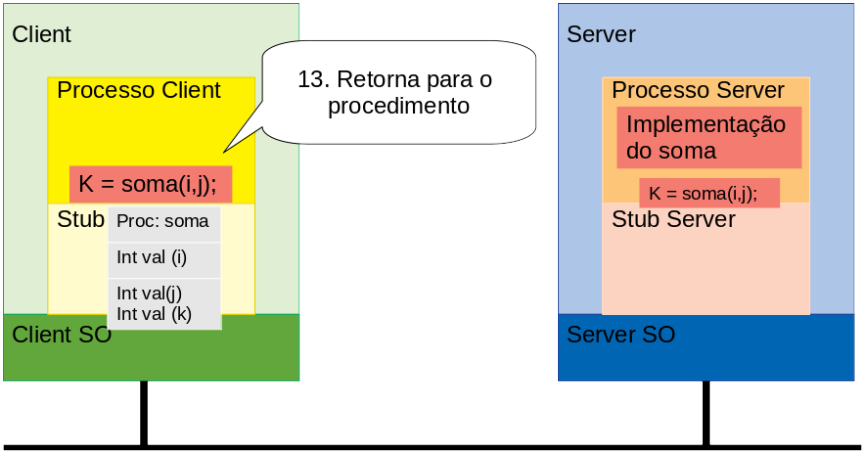
Chamada de Procedimento Remoto



Chamada de Procedimento Remoto



Chamada de Procedimento Remoto



Binding Dinâmico - RPC

- Para o cliente pedir para executar um método remoto, ele precisa saber **a localização do servidor**.
  - Uma alternativa é armazenar o endereço do servidor no programa (inflexível).
  - Outra alternativa é mapear cada serviço do servidor a uma porta específica de forma dinâmica:
    - *binding* dinâmico.

Binding Dinâmico - RPC

- O *binder* **funciona como um *nameserver*** para o RPC.
  - Ou seja, o cliente vai procurar um procedimento específico e o *binder* vai fornecer o endereço para o procedimento.
- É necessário o cliente enviar uma mensagem para o *binder* para localizar o servidor
- É possível o uso de múltiplos *binders*.

Binding Dinâmico - RPC

- Servidor
  - Ao ser executado, **avisa ao programa de registro** (*binder*) que está ativo: registro do serviço.
- Cliente
  - Quando um procedimento é chamado, este se liga ao servidor, **utilizando o *binder* como intermediário**.

Binding Dinâmico - RPC

- Vantagens:
  - Diversos servidores com a mesma “interface”;
  - Servidores que falham são automaticamente “desregistrados”;
  - Autenticação de usuários/clientes.
- Desvantagens:
  - *Overhead* de consulta ao *binder*;
  - Caso haja diversos *binders*, há a necessidade de atualização entre eles.

Falhas - RPC

- A concepção do RPC é deixar a programação transparente, mas as **seguintes falhas** podem ocorrer:
  - Cliente não acha o servidor;
  - A mensagem do cliente para o servidor foi perdida;
  - A mensagem do servidor para o cliente foi perdida;
  - O servidor sai do ar após receber uma solicitação;
  - O cliente sai do ar após ter enviado uma solicitação.

Falhas - RPC

- **Cliente não acha o servidor**:
  - O servidor está fora do ar;
  - Versões diferentes de stubs;
  - Soluções:
    - Retornos de variáveis “inválidas”: e.g. -1
    - Criação de exceções (perda de transparência).

Falhas - RPC

- **A mensagem do cliente para o servidor foi perdida**:
  - Limite de tempo de espera (*timeout*);
  - Reenvio em kernel;
  - Retorno de erro, após diversas tentativas.

Falhas - RPC

- **O servidor sai do ar após receber uma solicitação**:
  - Espera e reenvia / ache novo servidor;
  - Desiste e comunique falha.
    - As falhas até o momento não são distinguíveis para um cliente.

Falhas - RPC

- **O cliente sai do ar após ter enviado uma solicitação**:
  - Processamento órfão;
  - Gasto de tempo do servidor;
  - Soluções:
    - Reencarnação;
    - Extermínio;
    - Expiração (quantum T).

Falhas - RPC

- Extermínio:
  - Eliminar todos os órfãos.
  - Problema: encadeamento de falhas (servidor pode ter sido cliente em uma RPC).

Falhas - RPC

- Reencarnação:
  - Dividir o tempo em “épocas”.
  - O servidor pode checar a cada época chamadas de RPC sendo executadas por muito tempo.
    - Checar se o cliente da chamada ainda está ativo.
  - Se o cliente que fez a chamada caiu, o servidor pode enviar uma mensagem de *broadcast* para verificar se ele foi reiniciado.

Falhas - RPC

- Expiração:
  - Definir um tempo máximo para servidor executar serviço.
  - Problema: mensurar o quantum (unidade de tempo).

RPC - Remote Procedure Call

RMI - Java

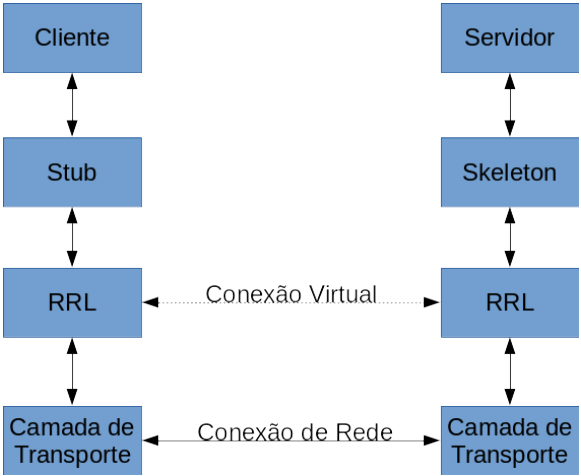
RMI - Java

- ***Remote Method Invocation***
  - RPC da linguagem Java;
  - **Suporta Orientação a Objetos**:
    - Permite o envio e recebimento de objetos.

RMI - Java

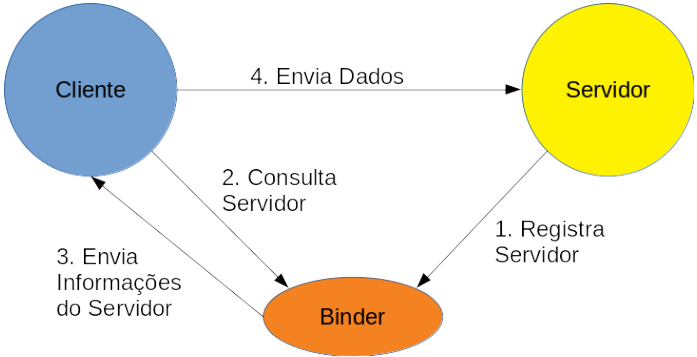
- Envio de mensagens realizado em 3 camadas:
  - *Stub/Skeleton*: é invocado primeiro, recebendo objetos e **serializando-os** (convertendo para bytes).
  - RRL – *Remote Reference Layer*: cada lado possui a sua, **responsável por montar a mensagem de envio/resposta entre cliente/servidor**.
  - Transporte: envio das informações via rede – TCP/IP.

RMI - Java


Cliente
Servidor
Stub
Skeleton
RRL
Conexão Virtual
RRL
Camada de Transporte
Conexão de Rede
Camada de Transporte

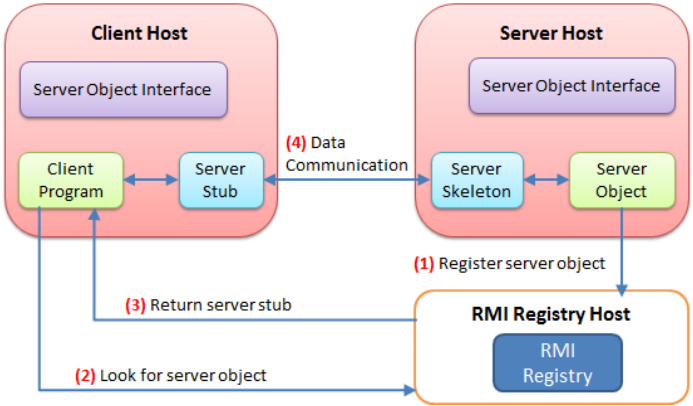
RMI - Java

- Processo de envio de mensagens:



RMI - Java

- Com mais detalhes:

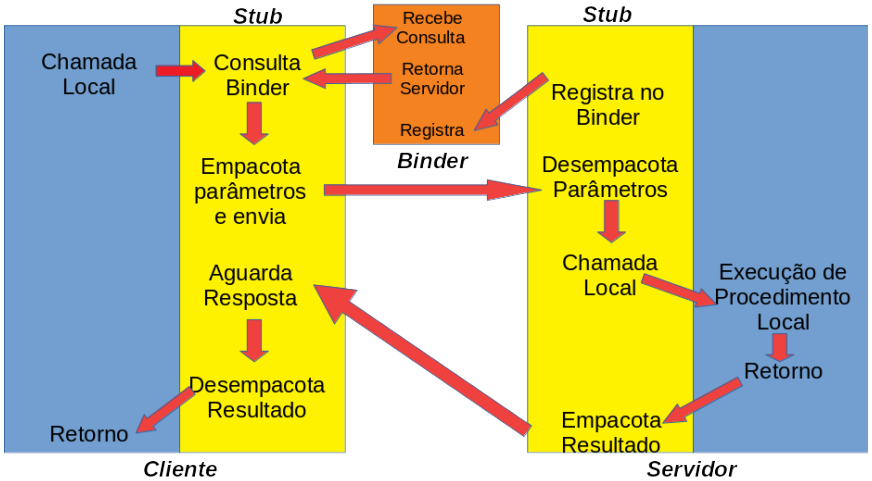


RMI - Java

1. O **`RMI Registry`** registra um novo objeto que pode ser procurado por clientes RMI.
2. O cliente procura pelo serviço específico.
3. O servidor **retorna um stub** para o cliente.
4. A comunicação entre cliente e servidor acontece, tendo os stubs como intermediários.

RMI - Java

- Ainda mais detalhes:



RMI

- Utilização de Objetos Java de forma “transparente”;
  - Uso de classe tipificada para a definição dos métodos:
    - `interface`.
  - A classe do cliente atua como proxy (ela só contém as assinaturas dos métodos);
  - **A implementação da classe fica no servidor**.

- Servidor:
  - Interface que contém os métodos a serem disponibilizados para os clientes.
- **implementa interface `Remote`**.
  - Classe que contém a lógica dos métodos, além das rotinas de registro.
- Herda características de **`UnicastRemoteObject`**.
- Implementa a interface definida.

Exemplo Servidor - RMI

- **Aplicação que possui um servidor que disponibiliza um método `soma()` para os clientes**.
  - Servidor se auto-registra no *binder*.
  - Cliente procura pelo objeto que possui esta funcionalidade (soma) através de seu nome.
  - Cliente executa o método correspondente.

Exemplo Servidor - RMI

- Baixe o arquivo:
  - `codigos_rpc.zip`

Exercício 1

- Altere o exemplo da calculadora para fazer as operações de soma, subtração, multiplicação e divisão, de acordo com a escolha do usuário.

Exercício 2

- Escreva uma aplicação dicionário.
  - O servidor conterá um arquivo com palavras em Português e Inglês separadas por vírgula.
    - Exemplo: sim,yes.
  - O Cliente irá chamar um método que recebe como parâmetro uma palavra em Português e retorna a tradução em Inglês.
  - Vamos fazer simplificações no dicionário, pois o conceito principal é a aplicação do RMI e não o dicionário propriamente dito.

Referências

- Livro:
  - Sistemas Distribuídos: Princípios e Paradigmas
    - Tanenbaum
    - Cap 10 - Sistemas distribuídos baseados em objetos
- Livro:
  - Sistemas Distribuídos: Conceitos e Projetos
    - Coulouris
    - Cap 5 - Invocação Remota

Conteúdo

https://gustavotcustodio.github.io/sdmobile.html

Obrigado

gustavo.custodio@ulife.com.br