

SISTEMAS DISTRIBUÍDOS

Professor: Johnatan Oliveira

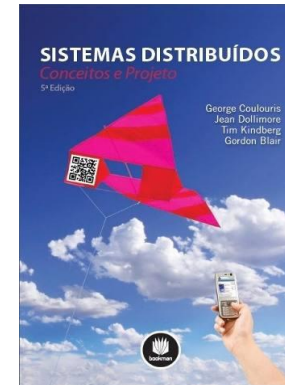
1

Material adaptado do professor Renato

SUMÁRIO

- Processos em Sistemas Distribuídos e Comunicação entre Processos
 - Virtualização
- Exercícios
- Invocação remota
 - Protocolos de Requisição-Resposta
 - RPC: Chamada a Procedimento Remoto;
 - RMI: Invocação a Métodos Remotos

Cap 5



VIRTUALIZAÇÃO

EXERCÍCIOS

- 1) Teria sentido limitar a quantidade de threads em um processo servidor?

EXERCÍCIOS

- 2) É concebivelmente útil que uma porta tenha vários receptores?

EXERCÍCIOS

- Aplicada em: 2014. Banca: CESPE. Órgão: TJ-SE
- Prova: Analista Judiciário - Suporte Técnico em Infraestrutura
- Com relação à virtualização, consolidação de servidores e *clusters* de alta disponibilidade, julgue os itens que se seguem.

A consolidação de servidores por meio da aplicação de técnicas e de ferramentas de virtualização permite economia nos custos operacionais e de aquisição da infraestrutura de tecnologia da informação.

- a) Certo
- b) Errado

EXERCÍCIOS

- Aplicada em: 2013. Banca: FUNCAB. Órgão: CODATA
- Prova: Auxiliar de Informática
- Analise as seguintes sentenças.
 - I. Na virtualização de servidores, existe um único sistema operacional e vários programas trabalhando em paralelo.
 - II. Existem várias ferramentas de virtualização no mercado e o VMware é uma delas.
 - III. A virtualização de *desktops* permite que as aplicações rodem em máquinas virtuais isoladas e ao mesmo tempo compartilhem recursos de *hardware* como CPU, memória, disco e rede.
- É(são) verdadeira(s), somente:
 - a) I
 - b) II
 - c) III
 - d) I e III
 - e) II e III

VIRTUALIZAÇÃO

- Threads e processos podem ser vistos como um modo de fazer diversas tarefas ao mesmo tempo
- Em computadores monoprocessados, execução simultânea é uma ilusão
 - único CPU, somente uma instrução de um única thread ou processo será executada por vez
- Virtualização de recursos: “fingir” que um determinado recurso esta replicado no sistema

VIRTUALIZAÇÃO

- Motivação atual:
 - Ajudar a transportar as interfaces herdadas para novas plataformas
 - Middleware e aplicações são muito mais estáveis que o hardware
 - Redes: reduzir a diversidade de plataformas e máquinas, em essência, deixando que cada aplicação execute em sua própria máquina virtual
 - Possivelmente incluindo as bibliotecas e o sistema operacional relacionados que, por sua vez, executam em uma plataforma comum
 - Alto grau de flexibilidade e portabilidade

VIRTUALIZAÇÃO

Sistemas de computadores oferecem 4 tipos diferentes de interface em 4 níveis diferentes:

- 1. Interface entre hardware e software para instruções de máquina invocadas por qualquer programa;
- 2. Interface entre hardware e software para instruções de máquina invocadas por programas privilegiados (S.O);
- 3. Interface de chamada de sistemas;
- 4. Interface de chamadas de bibliotecas (API).

VIRTUALIZAÇÃO

- **4 tipos de Interfaces diferentes em 4 níveis**

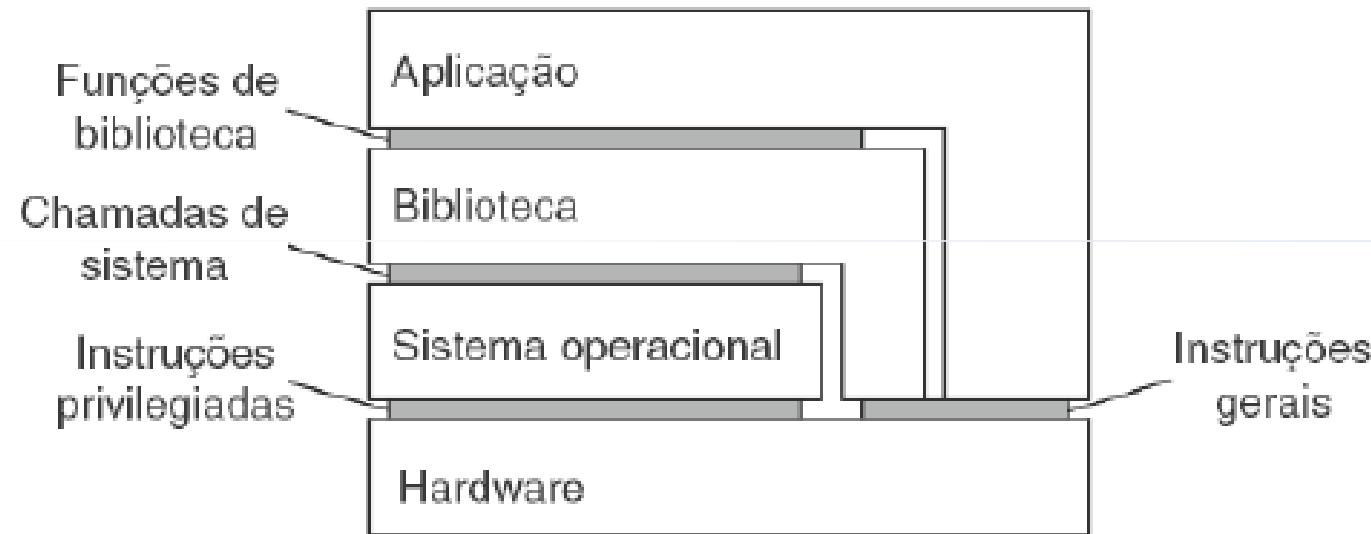


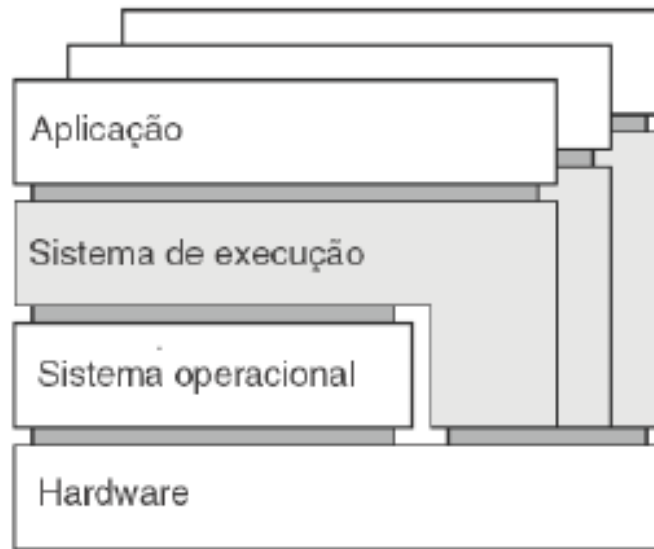
Figura 3.5 Várias interfaces oferecidas por sistemas de computadores.

- Essência da virtualização é imitar o comportamento das interfaces (instruções de máquina, chamadas de sistema)

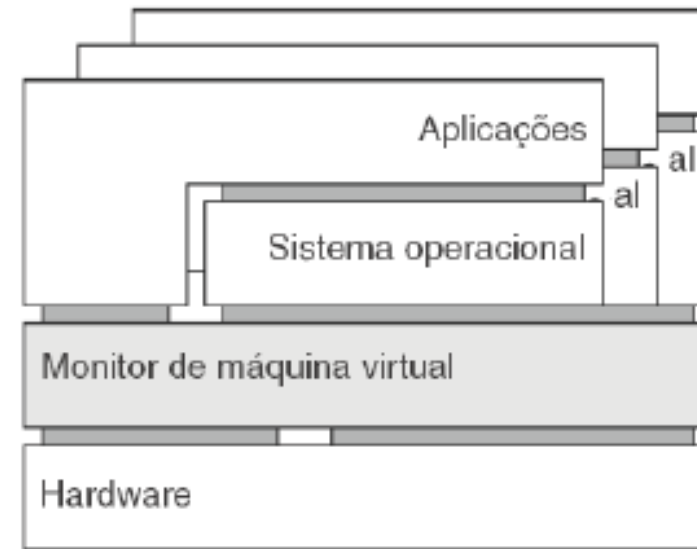
VIRTUALIZAÇÃO

- Arquiteturas de máquinas virtuais
 - **A virtualização pode ocorrer de 2 modos:**
 - **Máquina virtual de processo:**
 - Construir um sistema de execução que forneça um conjunto abstrato de instruções que deve ser usado para executar aplicações
 - Exemplo: linguagens interpretadas, como Java
 - **Fornecer um monitor de máquina virtual:**
 - É um sistema essencialmente implementado como uma camada que protege completamente o hardware mas oferece como interface o conjunto completo de instruções do mesmo;
 - Interface pode ser oferecida *simultaneamente* a programas diferentes;
 - Possível executar vários S.O.s simultaneamente e concorrentemente.
 - Exemplo: VMware

VIRTUALIZAÇÃO



(a)



(b)

Figura 3.6 (a) Máquina virtual de processo, com várias instâncias de combinações (aplicação, execução).
(b) Monitor de máquina virtual com várias instâncias de combinações (aplicações, sistema operacional).

INVOCÇÃO REMOTA

INTRODUÇÃO

- Como as Entidades se comunicam em um sistema distribuído?
- Pelo paradigma da invocação remota, geralmente temos três opções:
 - Protocolos de Requisição-Resposta;
 - suporte de nível relativamente baixo, para **RPC** e **RMI**
 - **RPC**: Chamada a Procedimento Remoto;
 - **RMI**: Invocação a Métodos Remotos.

MATÉRIA PARA A SEGUNDA ETAPA (2º PROVA)

Força e coragem faltam apenas 26 aulas para as férias

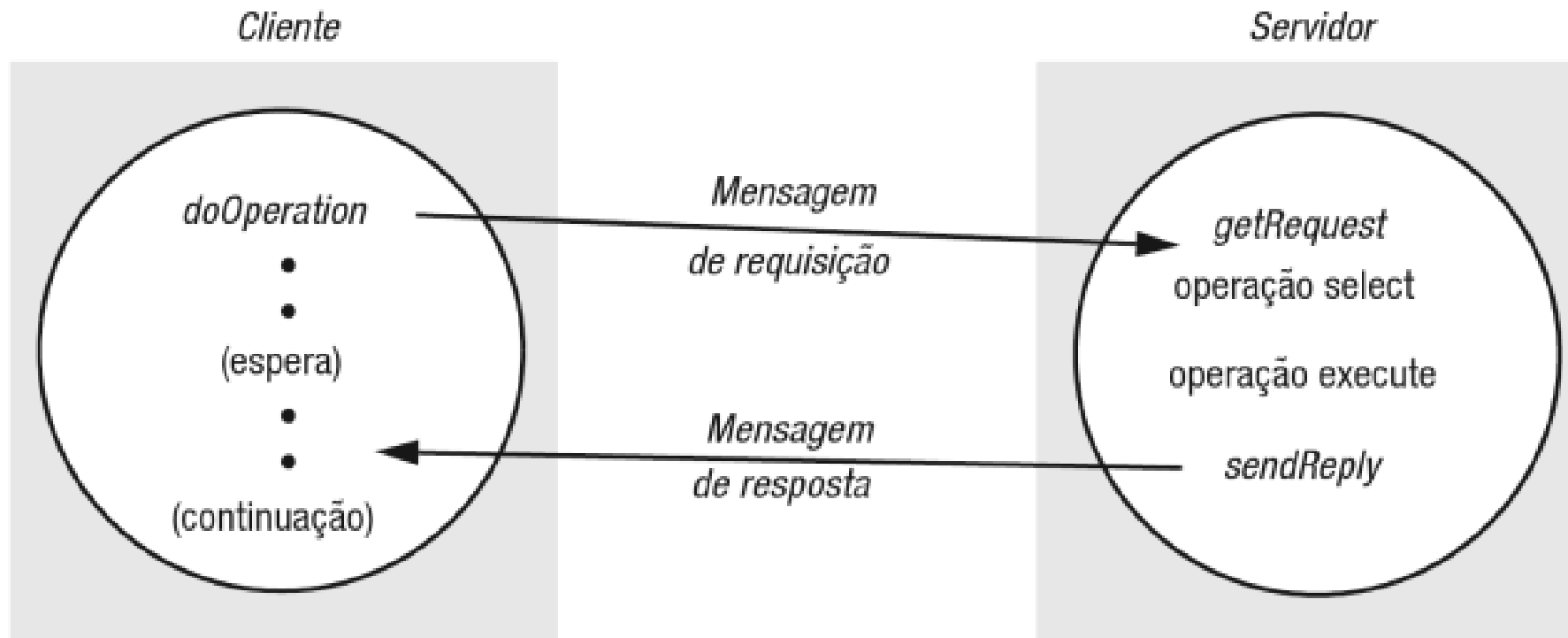


PROTOSCOLOS DE REQUISICÃO-RESPOSTA

- Forma de comunicação projetada para suportar as funções e as trocas de mensagens em interações **cliente e servidor**;
- Por padrão é uma comunicação **síncrona**;
 - Cliente fica bloqueado aguardando resposta
- Geralmente são implementadas sob **UDP**. Atualmente as implementações vem sendo feitas em **TCP**
 - Relembrem o uso de TCP e UDP
 - Como tornar uma transmissão por UDP confiável?
- Relembrem as operações *send* e *receive*;

PROTOS DE REQUISICÃO-RESPOSTA

- O protocolo que descrevemos aqui é baseado em um trio de primitivas de comunicação: `doOperation`, `getRequest` e `sendReply`;



PROTÓCOLOS DE REQUISIÇÃO-RESPOSTA

public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)

Envia uma mensagem de requisição para o servidor remoto e retorna a resposta.

Os argumentos especificam o servidor remoto, a operação a ser invocada e os argumentos dessa operação.

public byte[] getRequest ();

Lê uma requisição do cliente por meio da porta do servidor.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

Envia a mensagem de resposta *reply* para o cliente como seu endereço de Internet e porta.

PROTÓCOLOS DE REQUISIÇÃO-RESPOSTA

- **doOperation**
 - Usado pelos clientes para invocar operações remotas;
 - Seus argumentos especificam o servidor remoto, a operação a ser invocada e os argumentos da operação;
 - Seu resultado é um vetor de bytes contendo a resposta. O cliente que chama **doOperation** empacota os argumentos em um vetor de bytes e desempacota os resultados;
 - Após enviar a mensagem de requisição, este chama **receive** e fica bloqueado;
- **getRequest**
 - É usado por um processo servidor para obter as requisições de serviços;
- **sendReply**
 - Quando o servidor tiver invocado a operação especificada, ele usa **sendReply** para enviar a mensagem de resposta para o cliente. Quando a mensagem de resposta é recebida pelo cliente, **doOperation** é desbloqueada;

PROTÓCOLOS DE REQUISIÇÃO-RESPOSTA

- Implementado sobre datagrama UDP sofrerão de:
 - Falhas de omissão
 - Não há garantia de entrega das mensagens na ordem do envio
 - Falhas de colapso
 - O processo para e não produz nenhum resultado
- Tratamento: **timeout**
 - Ação executada quando o tempo estoura
 - Depende das garantias de entrega oferecidas
 - Ex. retransmitir mensagem
 - Mas isso pode fazer com que o servidor execute o mesmo processo duas ou mais vezes
 - Como tratar?
 - Identificação de mensagens e histórico

PROTOSCOLOS DE REQUISICÃO-RESPOSTA

- Estilos de protocolos de troca
 - R: request
 - RR: request-reply
 - RRA: request-reply-acknowledge reply
 - Servidor pode descartar histórico de mensagens confirmadas

| Nome | Mensagens enviadas pelo | | |
|------|-------------------------|----------|-------------------------|
| | Cliente | Servidor | Cliente |
| R | Requisição | | |
| RR | Requisição | Resposta | |
| RRA | Requisição | Resposta | Resposta de confirmação |

- Se o TCP for usado não precisa preocupar com confirmação

HTTP: EXEMPLO DE PROTOCOLO REQUISIÇÃO-RESPOSTA

- **HTTP** é usado pelos agentes de usuário para fazer pedidos para servidores Web e receber suas respostas;
- Os Servidores Web gerenciam recursos implementados de diferentes maneiras:
 - Como Dados: texto de uma página em HTML, uma imagem, um arquivo em pdf, etc;
 - Como um programa: ***servlets*** ou programas em PHP ou *Python*, que podem ser **executados no servidor**;
- A URL especifica o nome DNS de um servidor Web, um número de porta e o identificador de um recurso.
 - Porta padrão

HTTP: EXEMPLO DE PROTOCOLO REQUISIÇÃO-RESPOSTA

- O Protocolo HTTP especifica:
 - As mensagens envolvidas em uma troca de requisição-resposta;
 - Os métodos e os argumentos;
 - As regras para representá-los (empacotá-los) nas mensagens;
- Ele suporta um conjunto fixo de métodos (GET, PUT, POST, etc) que são aplicáveis a todos os recursos;

HTTP: EXEMPLO DE PROTOCOLO REQUISIÇÃO-RESPOSTA

- O Protocolo **HTTP** é implementado sob **TCP**;
- Na sua versão original, cada interação cliente-servidor consiste nas seguintes etapas:
 - O cliente solicita uma conexão com o servidor na porta HTTP padrão ou em uma porta especificada no URL;
 - O cliente envia uma mensagem de requisição para o servidor;
 - O servidor envia uma mensagem de resposta para o cliente;
 - A conexão é fechada;
- Atualmente o protocolo adota conexões persistentes;

MÉTODOS HTTP

- GET
 - Solicita o recurso cujo URL é dado como argumento;
- POST
 - Especifica o URL de um recurso (por exemplo, um programa) que pode tratar os dados fornecidos no corpo do pedido;
- DELETE
 - O Servidor exclui o recurso especificado na URL;
- PUT
 - Solicita que os dados fornecidos na requisição sejam armazenados no URL informado (Atualização);
- HEAD
 - Retorna informações sobre um recurso. Na prática, funciona semelhante ao método GET, mas sem retornar o recurso no corpo da requisição.

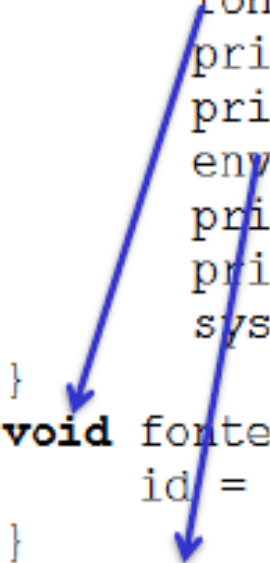
CHAMADA DE PROCEDIMENTO REMOTO (RPC)

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- **Objetivo:** tornar a programação de sistemas distribuídos semelhante (se não idêntica) à programação convencional
 - Obter transparência de distribuição de alto nível
- Estende a abstração de chamada de procedimento para os ambientes distribuídos
 - Possível chamar procedimentos remotos como se estivessem no endereçamento local
 - Oculta aspectos da distribuição
 - Codificação
 - Decodificação
 - Passagem de mensagens
 - ...

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

```
main() {  
    char nome[5];  
    int idade = 31;  
  
    printf("Informe seu nome:");  
    gets(nome);  
    printf("Informe sua idade:");  
    scanf("%d", &idade);  
    fontedaJuventude(idade);  
    printf("Nome:%s\n", nome);  
    printf("Idade:%d\n", idade);  
    envelhecer(&idade);  
    printf("Nome:%s\n", nome);  
    printf("Idade:%d\n", idade);  
    system("pause");  
}  
  
void fontedaJuventude(int id) {  
    id = id + 1;  
}  
  
void envelhecer(int *id) {  
    *id = *id + 1;  
}
```

Two blue arrows originate from the function calls in the main function. The first arrow points from 'fontedaJuventude(idade);' to the definition of 'fontedaJuventude(int id)'. The second arrow points from 'envelhecer(&idade);' to the definition of 'envelhecer(int *id)'.

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- Antes de examinarmos a implementação em sistemas de RPC, vamos ver 3 questões importantes para entender esse conceito:
 - O estilo de programação promovido pela RPC: programação com interfaces;
 - A semântica de chamada associada à RPC;
 - O problema da transparência e como ele se relaciona com as chamadas de procedimento remoto;

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- Programação com Interfaces
- As modernas linguagem de programação promovem a modularização e comunicação entre estes módulos;
- Para controlar as possíveis interações entre os módulos, é definida uma *interface* que especifica os procedimentos e as variáveis que podem ser acessadas a partir de outros módulos;
- Contanto que sua interface permaneça a mesma, a implementação pode ser alterada sem afetar os usuários do módulo
- Em um modelo cliente-servidor geralmente surge o termo, *interface de serviço*: se refere a especificação dos procedimento oferecidos por um servidor, definindo os tipos de argumentos de cada um dos procedimentos;

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- Programação com Interfaces
- Vantagens na programação de interfaces nos SDs:
 - Os programadores só se preocupam com a abstração oferecida pela interface
 - Desconhece detalhes da implementação
 - Programadores não precisam conhecer a linguagem de programação nem a plataforma de base utilizadas para implementar o serviço
 - Evolução do software
 - Implementações podem mudar desde que a interface permaneça a mesma

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- Programação com Interfaces
- Definição de interface é influenciada pela natureza distribuída da infraestrutura subjacente:
 - Não é possível um módulo ser executado em um processo e acessar as variáveis de um módulo em outro processo
 - Usa procedimentos getter e setter
 - Chamada por referência não são suportadas
 - Parâmetros são de entrada ou de saída e devem ser transmitidos nas mensagens
 - Os endereçamentos de um processo não são válidos em outro processo remoto
 - Endereços não podem ser passados/retornados como argumentos

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- Uso de Linguagens de Definição de Interfaces:
 - As **IDLs** são projetadas para permitir que procedimentos implementados em diferentes linguagens invoquem uns aos outros;
 - Ex.: serviços escritos em C++ podem ser acessados remotamente pela linguagem Java
 - IDL fornece uma notação para definir interfaces
 - Cada parâmetro é especificado quanto ao tipo e se é de entrada ou saída
 - Conceito desenvolvido inicialmente para RPC mas também se aplica à RMI e serviços Web

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- **Ex. de IDL do CORBA**

// Arquivo de entrada Person.idl

*struct Person {
 string name;
 string place;
 long year;*

};

interface PersonList {
 readonly attribute string listname;
 void addPerson(in Person p) ;
 void getPerson(in string name, out Person p);
 long number();

Métodos disponíveis em
um objeto remoto



};

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- Semântica de chamada RPC
- Promove a confiabilidade das invocações remotas vistas pelo ativador/chamador;
- Semântica talvez: a chamada de RPC pode ser executada uma vez ou não ser executada. Ele surge quando nenhuma medida de tolerância a falhas é aplicada;
- Semântica pelo menos uma vez: o invocador recebe um resultado (o procedimento foi executado pelo menos uma vez) ou recebe uma exceção;
- Semântica no máximo uma vez: o chamador recebe um resultado – no caso em que o chamador tem certeza da execução – ou recebe uma exceção informando-o de que nenhum resultado foi recebido.

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- **Semântica de chamada RPC**

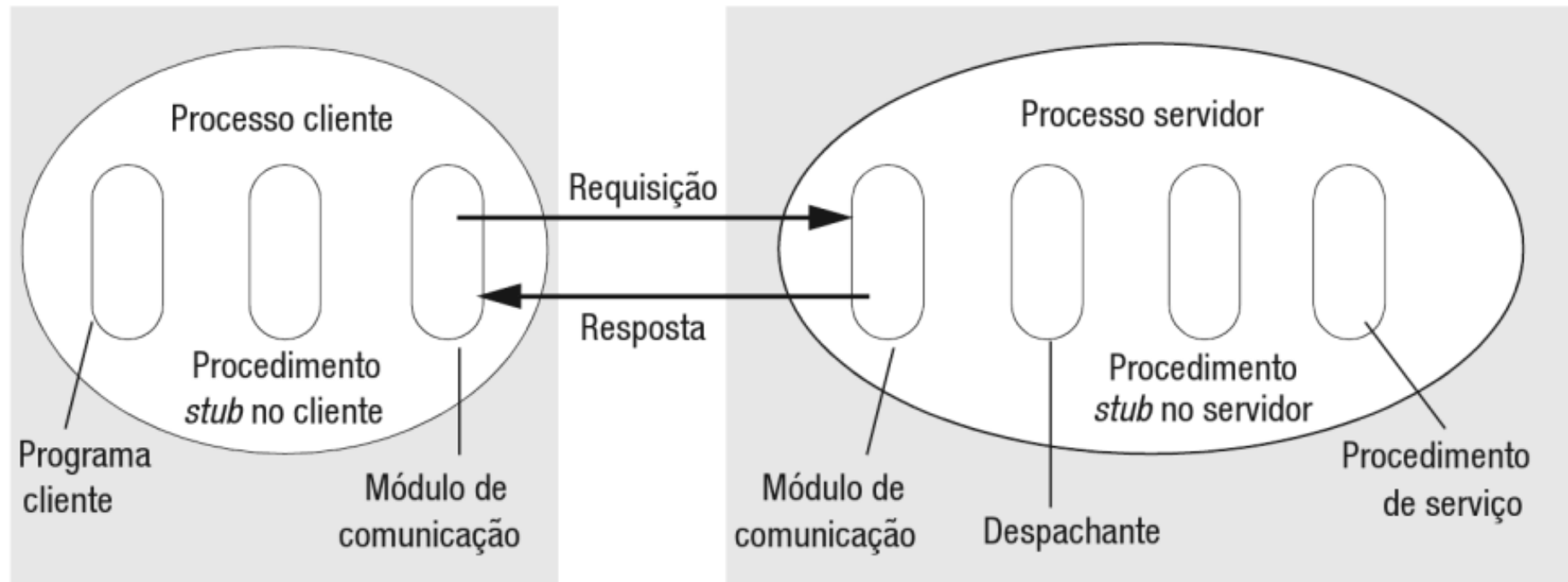
| Medidas de tolerância a falhas | | | Semântica de chamada |
|-----------------------------------|-------------------------|---|---------------------------|
| Reenvio da mensagem de requisição | Filtragem de duplicatas | Reexecução de procedimento ou retransmissão da resposta | |
| Não | Não aplicável | Não aplicável | <i>Talvez</i> |
| Sim | Não | Executa o procedimento novamente | <i>Pelo menos uma vez</i> |
| Sim | Sim | Retransmite a resposta | <i>No máximo uma vez</i> |

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- **Transparência**
- Objetivo Básico do RPC: Chamada Local = Chamada Remota;
- Uma série de ações transparentes ao Programador (empacotar/desempacotar, trocas de mensagens, etc);
- Transparência de localização e de acesso, ocultando o local físico do procedimento;
- Perigoso: Chamadas de Procedimentos Remotos são mais vulneráveis a falhas (envolvem rede, outro computador, outro processo);
 - Questões de desempenho/segurança são afetadas
- Consenso atual: Chamadas locais e remotas devem se tornar transparente em sintaxe de uma chamada local ser igual a uma remota, mas a diferença entre as duas devem ser expressas em suas interfaces;

CHAMADA DE PROCEDIMENTO REMOTO (RPC)

- Implementação:
 - Componentes de software exigidos para implementar RPC
 - Stub: interface entre o cliente e o servidor

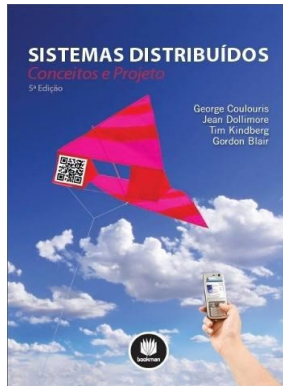


CHAMADA DE PROCEDIMENTO REMOTO (RPC)

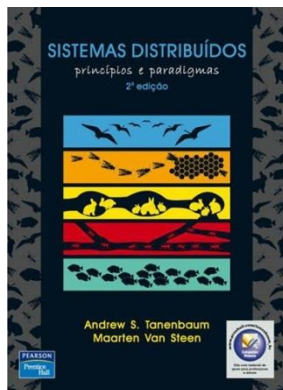
Implementação

- Cliente mantém um *procedimento stub* para cada procedimento da interface de serviço;
- O *procedimento stub* se comporta como local para o cliente;
- Na realidade o *procedimento stub*: empacota o identificador de procedimento e os argumentos e os envia através do módulo de comunicação;
- O Servidor mantém um despachante que seleciona um dos *procedimentos stub* do servidor (identificador de procedimento);
- O *procedimento stub* no servidor desempacota os argumentos da mensagem de requisição e chama o procedimento de serviço;
- Após a execução, os valores de retorno são empacotados e enviados ao cliente;
- Os procedimentos de serviço implementam os procedimentos da interface de serviço.

LIVRO TEXTO



- (Coulouris, 2013)
COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.
Sistemas Distribuídos: Conceito e Projeto
Artmed, 5ª edição, 2013



- (Tanenbaum, 2008)
TANENBAUM, A. S.; STEEN, M. V.
Sistemas Distribuídos
Pearson, 2ª edição, 2008