SSC150 – Sistemas Computacionais Distribuídos

Comunicação em Sistemas Distribuídos
RPC

4ª aula
25/03/10

Profa. Sarita Mazzini Bruschi
sarita@icmc.usp.br

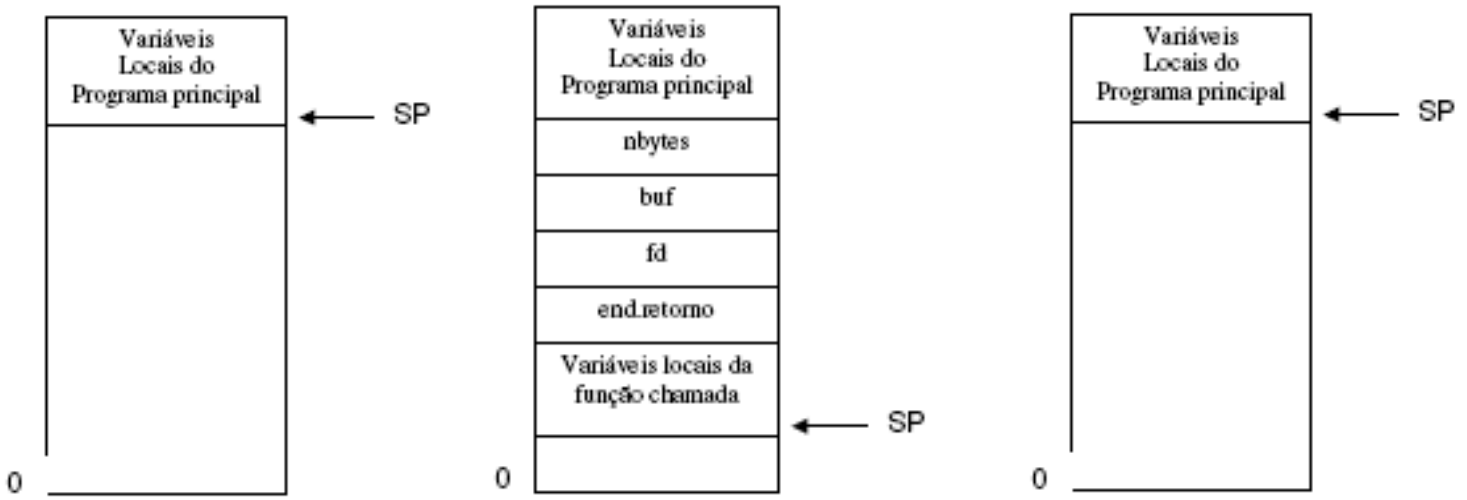
Slides baseados no material de:
Prof. Rodrigo Mello (USP / ICMC)
Prof. Edmilson Marmo Moreira (UNIFEI / IESTI)

Chamada Remota de Procedimentos Remote Procedure Call (RPC)

- Problema do modelo Cliente – Servidor:
  - A comunicação é realizada através das primitivas *send* e *receive*, as quais são consideradas como Entrada/Saída
  - Entrada/Saída não é o conceito chave para sistemas centralizados
  - Perda de transparência, pois a idéia é ter impressão de um sistema centralizado
- Idéia básica do RPC:
  - programas (ou processos) podem chamar procedimentos localizados em outras máquinas, sem declarar explicitamente as funções *send* e *receive*
  - Idéia simples e elegante, fazendo a chamada remota se parecer o máximo possível com a chamada local
  - Problemas: quando as máquinas são diferentes (espaço de endereçamento, parâmetros diferentes) e quando há falhas em uma das máquinas

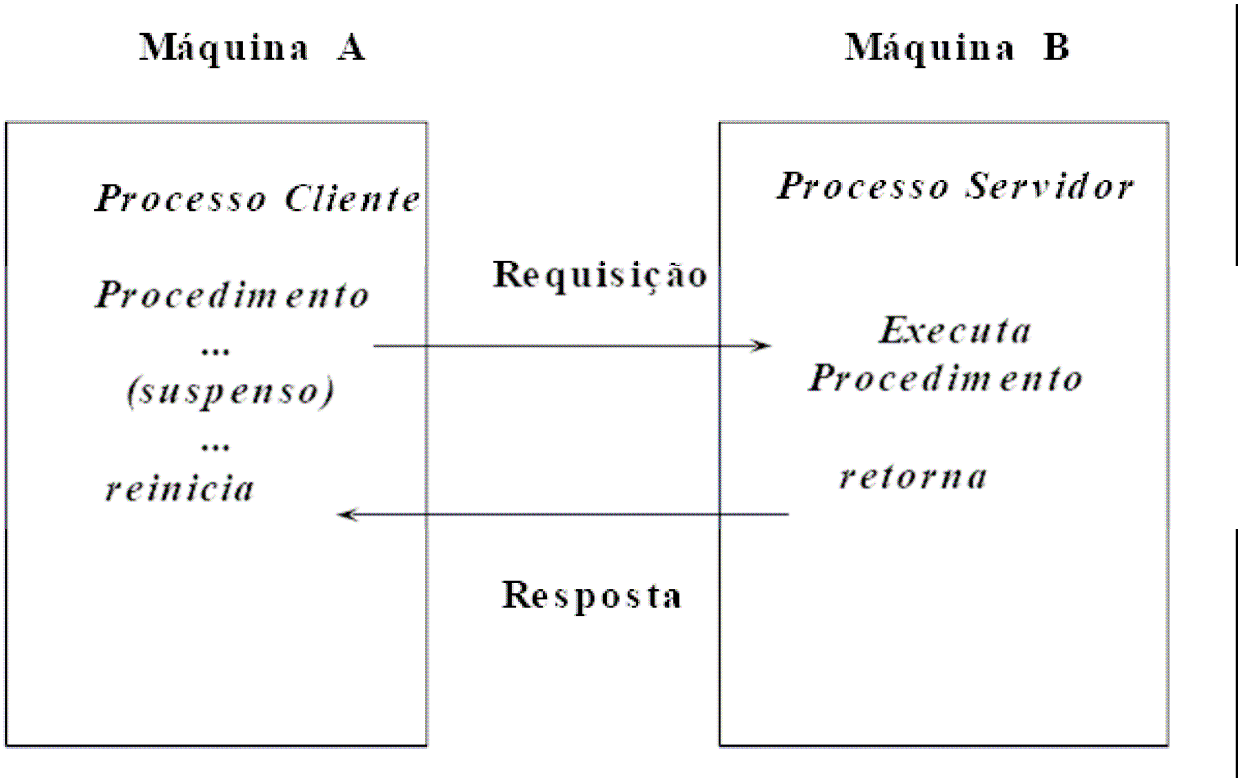
Chamada Local

- Exemplo de uma pilha em uma chamada local:
  - count = read(fd, buf, nbytes)

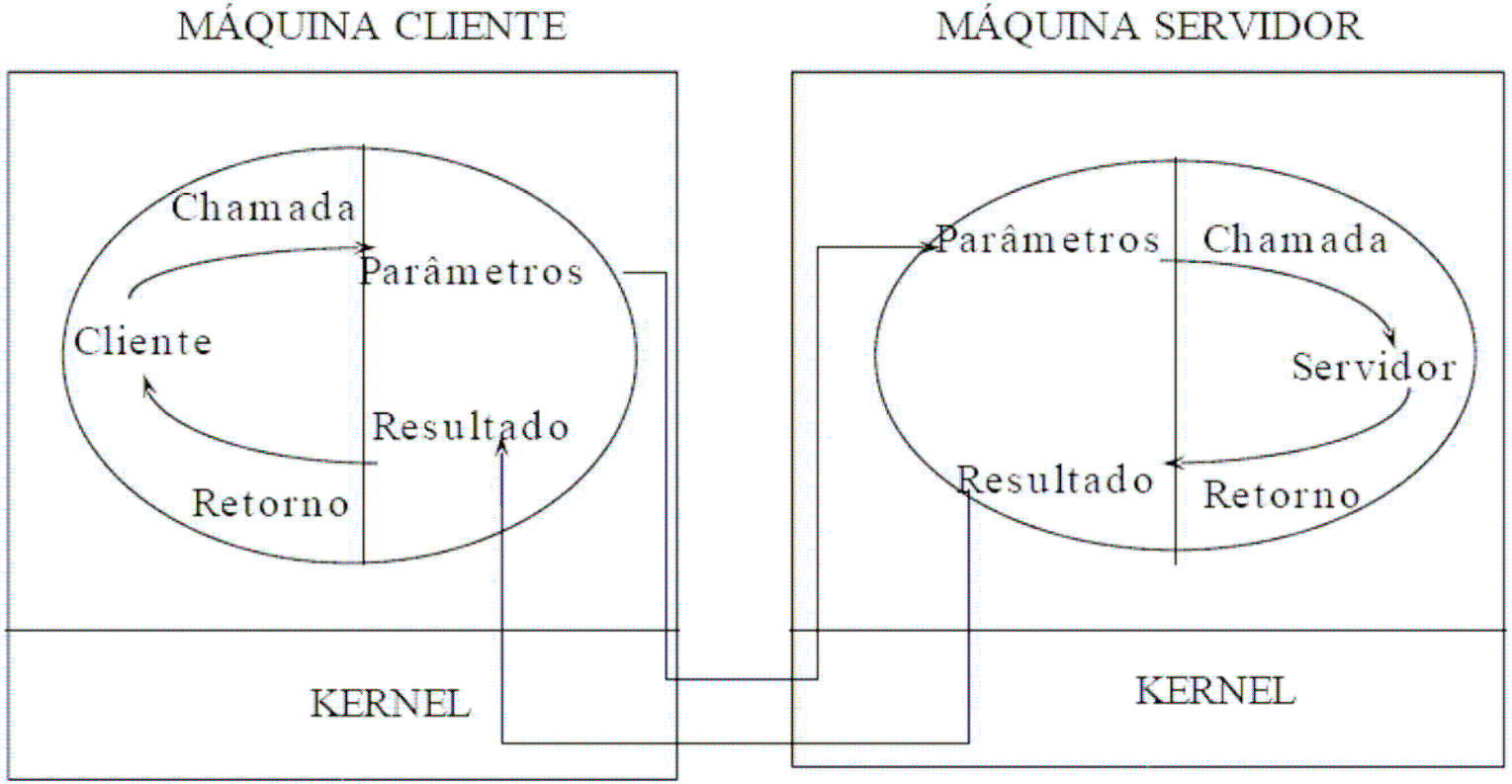


Antes da chamada | Durante a chamada | Após a chamada

Idéia básica da chamada remota



Implementação da chamada remota



Implementação da chamada remota

- Quando a chamada é remota uma versão diferente do procedimento chamado é usada (*client stub*).
- Ao invés de colocar os parâmetros na pilha (como na chamada local), a chamada remota pede ao kernel que envie uma mensagem com os parâmetros para o servidor. O processo cliente fica bloqueado até receber a resposta com o resultado da chamada remota
- Do lado do servidor, quando a mensagem chega com o pedido de execução remota, transfere o pedido para uma versão diferente do servidor (*server stub*).
- Os parâmetros são disponibilizados e o procedimento servidor é chamado no modo usual (chamada local). Quando o servidor remoto assume o controle novamente depois da chamada ter completado, os resultados são enviados para o cliente

Implementação da chamada remota

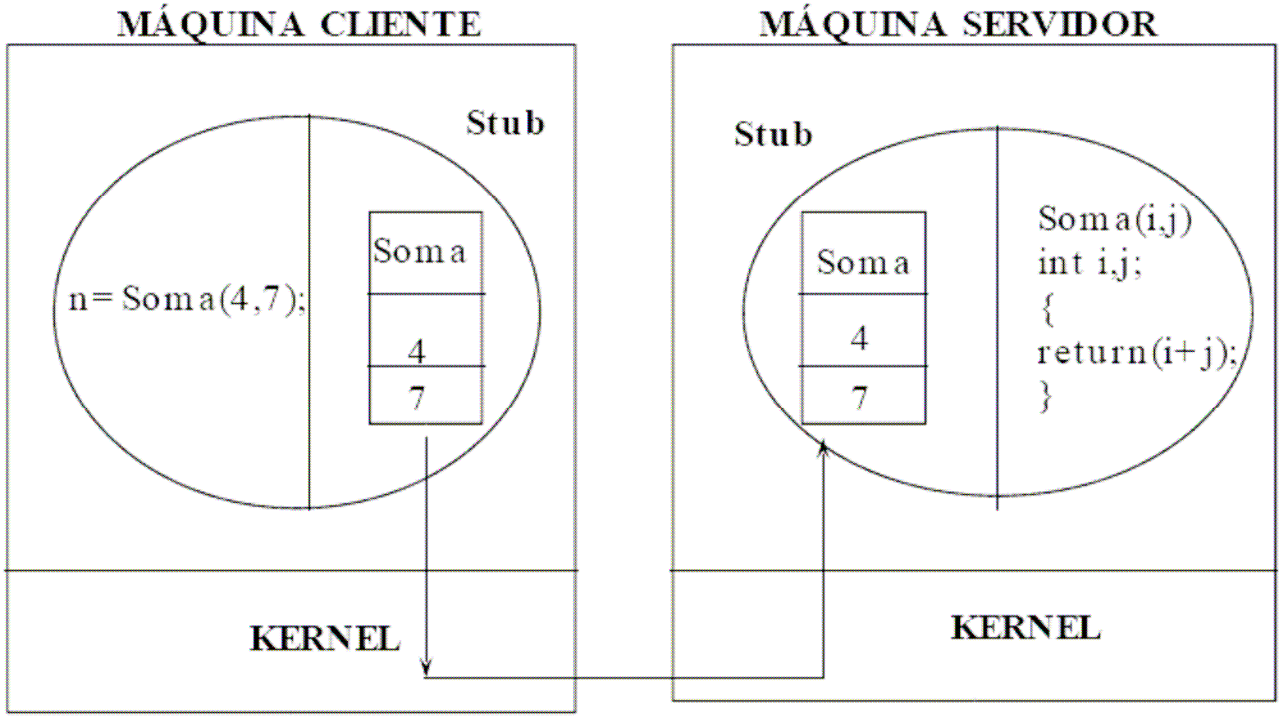
- Quando a mensagem de resposta chega ao cliente, o kernel identifica o endereço como sendo o do processo cliente. A mensagem é copiada no buffer e o cliente desbloqueado.
- A versão remota do cliente copia o resultado para o cliente local e retorna ao funcionamento usual. Quando o cliente local ganha o controle ele sabe que o dado está disponível, mas não tem idéia de onde ele foi processado

Passagem de parâmetros

- Tem três modos de se passar parâmetros em um procedimento:
  - passagem por Valor ;
  - passagem por Referência;
  - passagem por Cópia/Restaura
- Função do Cliente "*stub*":
  - Pegar os parâmetros, colocá-los na mensagem e enviar ao Servidor "*stub*"

Exemplo

- Soma(i,j); sendo i e j inteiros

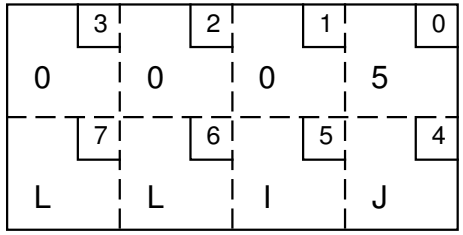


Passagem de parâmetros

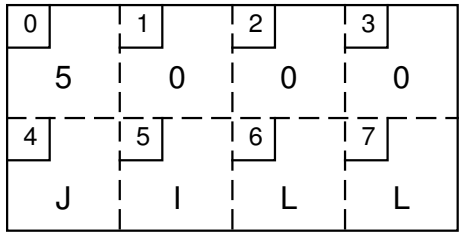
- Problema: em um grande Sistema Distribuído, normalmente existem vários tipos dé máquinas
- Exemplo:
  - IBM mainframes: EBCDIC
  - IBM PCs: ASCII
  - Intel 486: numera os bytes de um inteiro da direita para a esquerda (little endian)
  - Sun SPARC: na ordem inversa (big endian)

Passagem de parâmetros

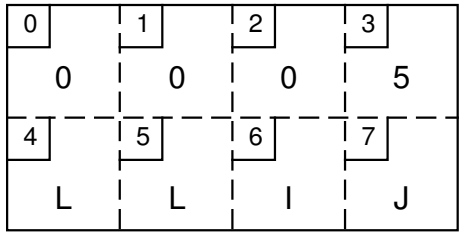
Mensagem de 32 bits
2 parâmetros:
  um inteiro (5)
  uma string (JILL)



Mensagem original no 486 (numera os bits de um inteiro da direita para a esquerda – Little endian)



Mensagem recebida na SPARC (na ordem inversa – Big endian)



Mensagem invertida

Passagem de parâmetros

- A simples inversão dos bytes de cada palavra depois de recebido não produz o resultado correto:
  - strings não são colocadas na forma reversa
  - a inversão deveria ser apenas para inteiros
- Solução:
  - Tanto o cliente como o servidor conhecem o tipo dos parâmetros e pelo tipo é possível saber os que devem ser invertidos
  - E quando isso não é conhecido?

Passagem de parâmetros

- Outra solução:
  - Criar um padrão de rede, definindo o formato dos inteiros, caracteres, ponto flutuante, etc.
  - Requer que os dados sejam convertidos para esse padrão antes de serem enviados, técnica conhecida como *marshalling*
  - Problema:
    - Ineficiência: máquinas com a mesma representação farão duas conversões quando na realidade não é necessário fazer nenhuma
- Outra solução:
  - O primeiro byte da mensagem indica qual é o formato usado
  - A conversão é realizada somente quando os formatos forem diferentes

Passagem de parâmetros

- Geração dos stubs:
  - Geralmente os procedimentos *stubs* são gerados automaticamente a partir de uma única especificação formal do servidor
  - Isso facilita a vida do programador, reduz a possibilidade de erros e torna o sistema transparente
- Como tratar os ponteiros?
  - Solução 1: proibir o uso de ponteiros e passagem de parâmetros por referência

Passagem de parâmetros

- Solução 2:
  - Copiar o arranjo apontado pelo ponteiro na mensagem e enviá-lo ao servidor
  - O servidor *stub* pode então chamar o servidor com um ponteiro para esse arranjo
  - As mudanças que o servidor fizer nas posições apontadas pelo ponteiro afetam o buffer da mensagem do servidor *stub*
  - Quando o servidor termina, o arranjo é enviado de volta para o cliente *stub*, que copia de volta ao cliente
  - Efeito: a chamada por referência foi substituida pela chamada cópia/restaura

Binding dinâmico

- Como o cliente localiza o servidor?
  - Uma solução é anexar o endereço do servidor no cliente
  - Inflexível
- Solução: *binding* dinâmico
  - Utiliza uma especificação formal dos servidores, composta pelo nome do servidor, versão e lista de procedimentos oferecidos pelo servidor
  - A especificação formal é utilizada como entrada do gerador de *stubs*, e produz o *stub* cliente e o *stub* servidor, os quais são colocados dentro das bibliotecas apropriadas
  - Quando um programa do usuário cliente chama algum destes procedimentos, o procedimento do cliente *stub* correspondente é linkado com o seu binário

Binding dinâmico
Especificação formal

```
#include <header.h>
// specification of file_server, version 3.1:
long read (in  char name[MAX_PATH], out char buf
        [BUF_SIZE], in long bytes, in long position);

long write (in char name[MAX_PATH], in char buf[BUF_SIZE],in
long bytes, in long position);

int create (in char[MAX_PATH], in int mode);

int delete (in char[MAX_PATH]);
end;
```

Binding dinâmico

- Quando o servidor começa a executar, ele chama a função que inicializa (***initialize***), a qual exporta (***export***) a interface do servidor, isto é, envia uma mensagem para o programa *binder* para ser tornar conhecido
- Esse processo é chamado de registro do servidor

Binding dinâmico

- Interface do *binder*

Chamada	**Entrada**	**Saída**
Registro	Nome, versão, *handle*, id	
Desregistro	Nome, versão, id	
Consulta	Nome, versão	*handle*, id

Binding dinâmico

- Como o cliente localiza o servidor?
  - Quando um cliente faz uma chamada remota pela primeira vez, o cliente *stub* vê que o serviço não está ligado a um servidor
  - Desse modo, ele envia uma mensagem para o *binder* pedindo para importar (***import***) a versão 3.1 do file_server
  - O *binder* verifica se algum servidor exportou uma interface com esse nome e versão. Em caso negativo, a chamado remota irá falhar

Binding dinâmico

- Método para importar e exportar interfaces é altamente flexível:
  - Múltiplos servidores com a mesma interface;
  - Servidores que falham em responder são “desregistrados”
  - Desvantagens de *overhead* extra, pois clientes devem receber interfaces toda vez que inicializarem comunicação (sobrecarga da rede)
  - Grandes SDs necessitam de múltiplos *binders*
    - pode-se replicar *binders*, porém a sincronização não é simples: requer mensagem de registro, desregistro, etc. atuando sobre todos

Presença de falhas em RPC

- Possíveis falhas em RPC:
  1. O cliente não consegue localizar o servidor
  2. A mensagem de requisição do cliente para o servidor é perdida
  3. A mensagem de resposta do servidor para o cliente é perdida
  4. O servidor falha depois de receber uma requisição
  5. O cliente falha depois de enviar uma requisição

Presença de falhas em RPC
Cliente não consegue localizar o servidor

- Causas:
  - Servidor desligado;
  - Versão desatualizada da interface
- Soluções:
  - Retornar código indicando erro;
  - Ativar exceção
    - escrever um procedimento de tratamento de exceção destrói a transparência

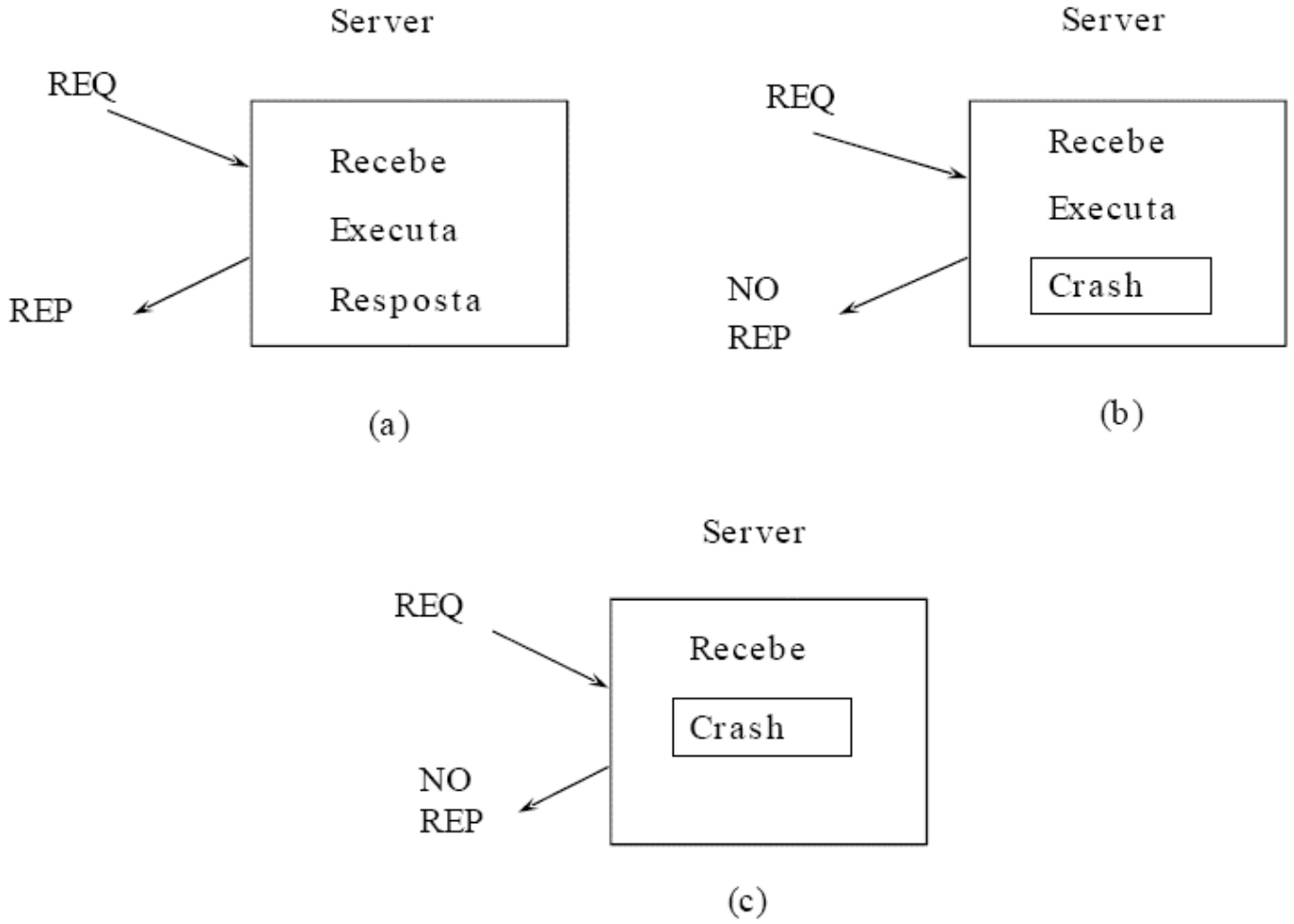
Presença de falhas em RPC
Mensagem cliente-servidor perdida

- O kernel do cliente inicializa um *timer* assim que a mensagem é enviada
- Se o *timer* expirar antes de receber uma resposta ou um *acknowledgement*, o kernel envia a mensagem novamente

Presença de falhas em RPC
Mensagem servidor-cliente perdida

- Solução óbvia: *timer*
  - Se a resposta não chegar após um determinado tempo, envia a requisição novamente
- Problema:
  - O kernel do cliente não sabe ao certo porque não recebeu resposta
  - O cliente pode perguntar:
    - A requisição ou a resposta foi perdida?
    - Ou o servidor está lento?
  - Algumas requisições podem ser repetidas sem maiores problemas (leitura de arquivos), porém outras não (transações bancárias)
  - Solução:
    - o kernel do cliente atribui a cada requisição do cliente um número de seqüência
    - o servidor saberá distinguir entre uma requisição já respondida e uma ainda não processada

Presença de falhas em RPC
Falha no servidor



Presença de falhas em RPC
Falha no servidor

- Tratamento diferente entre as falhas (b) e (c)
  - em (b) o sistema tem que comunicar a falha para o cliente
  - em (c) precisa somente retransmitir a mensagem
- Existem 4 semânticas:
  - *At least once*
  - *At most once*
  - *Maybe*
  - *Exactly once*

Presença de falhas em RPC
Falha no servidor

- *At least once*
  - Espera que o servidor seja reinicializado e tenta novamente
  - Objetivo é tentar até que a mensagem de resposta seja recebida
  - Garante que a chamada remota seja executada pelo menos uma vez, mas possivelmente muitas

Presença de falhas em RPC
Falha no servidor

- *At most once*
  - Informa imediatamente uma falha quando ela ocorre
  - Garante que a chamada remota foi executada no máximo uma vez, mas possivelmente nenhuma
- *Maybe*
  - Não garante nada
- *Exactly once*
  - Ideal

Presença de falhas em RPC
Falha no cliente

- O que acontece se um cliente envia uma requisição para um servidor e falha antes de receber a resposta?
- Teremos uma computação ativa sem um processo "pai" esperando pelo resultado. Esse tipo de computação não deseja é denominada ÓRFÃO
- Problemas com órfãos:
  - Desperdício de CPU
  - Podem segurar recursos (*deadlock*)
  - Quando o cliente é reinicializado, a chamada remota é executada novamente e logo em seguida vem o resultado órfão... Pode haver confusão dos resultados

Presença de falhas em RPC
Falha no cliente

- Solução 1:
  - Cliente *stub* mantém um registro sobre todas as mensagens RPC.
  - O registro é mantido em disco ou outro meio que sobreviva a uma falha (persistente). Depois que o cliente reinicializa, o registro é verificado e os órfãos são eliminados. Esta solução é chamada ***exterminação***
  - Problemas:
    - *Overhead* escrevendo o registro no disco;
    - Pode não funcionar se o órfão fez uma chamada remota (*grandorphans*)
      - se houver uma falha na rede pode ser impossível eliminar o órfão mesmo sabendo onde ele se encontra

Presença de falhas em RPC
Falha no cliente

- Solução 2 – ***Reencarnação***
  - O tempo é dividido em épocas numeradas sequencialmente. Quando o cliente é reinicializado ele envia uma mensagem para todas as máquinas declarando o começo de uma nova época (*broadcast*). Quando mensagem é recebida todas as chamadas remotas da "época" passada são eliminadas
  - Se existirem redes particionadas, ao receber uma resposta o cliente checa no cabeçalho a época e desconsidera a mensagem se for de uma época anterior

Presença de falhas em RPC
Falha no cliente

- Solução 3: ***Reencarnação suave***
  - Quando a mensagem de uma nova "época" é recebida, cada máquina servidora verifica se tem computações remotas; em caso positivo tenta localizar seu cliente. Se ele não for encontrado a computação é eliminada
- Solução 4: ***Expiração***
  - Cada RPC recebe uma quantidade de tempo padrão T para executar seu trabalho. Depois de uma falha a reinicialização é feita após um tempo T, quando os órfãos terão terminado
  - Problema: escolher um valor para T (RPC tem diferentes requerimentos)