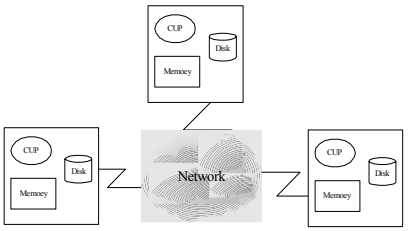
SISTEMAS DISTRIBUÍDOS



Comunicação em Sistemas Distribuídos

Sumário

- Modelo Cliente e Servidor
- Troca de Mensagens
- *Remote Procedure Call*
- Comunicação Grupal
- Objetos Distribuídos
- Outros Mecanismo de Comunicação

Comunicação em Sistemas Distribuídos

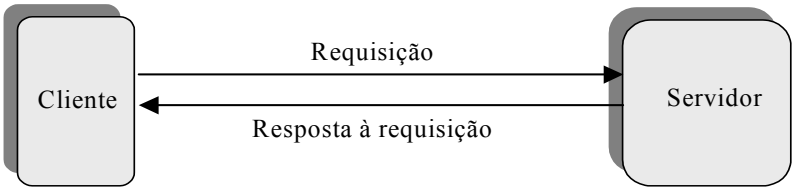
- Quando múltiplos processos fazem um trabalho conjunto, eles devem interagir
- "Comunicação interprocesso" (IPC): forma de interação ou comunicação entre processos
- Sistemas distribuídos possuem mecanismo de comunicação entre processos em diferentes máquinas (remotas), ***pois não há compartilhamento de memória física***

Modelo Cliente/Servidor

- Base para um sistema distribuídos
- É um processamento cooperativo de requisições submetidas por um cliente a um servidor que as processa e retorna um resultado
- É uma forma especial de processamento distribuído em que os recursos estão espalhados em mais de um computador.

Modelo Cliente/Servidor

- Protocolo
  - Request (Requisição)
  - Reply (Resposta)
  - Simples e direto



Modelo Cliente/Servidor

- Cliente/Servidor – Primeira Geração
  - Modelo de processamento para compartilhamento de dispositivos
  - Modelo de processamento cliente/servidor
  - Processamento Peer-to-Peer (Igualitário ou ponto-a-ponto)
- Cliente/Servidor – Segunda Geração
  - Evolução do sistema duas camadas para um sistema com várias camadas, altamente distribuído e cooperativo;

Implementação Cliente/Servidor

- Baseada no protocolo de Request (Requisição) e Reply (Resposta)
- Utiliza troca de mensagens através da rede
- Questões a tratar:
  - Endereçamento
  - Primitivas empregadas
  - Bufferização
  - Confiabilidade

Implementação Cliente/Servidor
(Endereçamento)

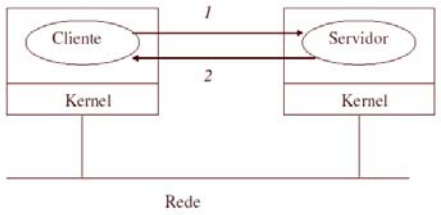
- Um cliente para mandar uma mensagem a um servidor precisa saber o endereço
- Existem várias formas de endereçamento
- Principais destacados:
  - Endereçamento por número de máquina
  - Endereçamento por processo
  - Endereçamento por nomes ASCII obtidos de um servidor de nomes (name server)

Implementação Cliente/Servidor
(Endereçamento)

- Endereçamento por número de máquina

um processo por máquina
não é transparente
simplicidade

1. *Request* para 243
2. *Replay* para 199

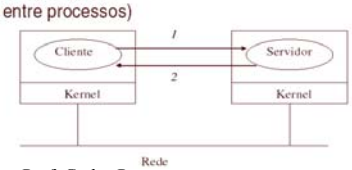


Implementação Cliente/Servidor
(Endereçamento)

- Endereçamento por número de máquina
  - Pode-se utilizar uma combinação entre número da máquina e número do processo

não é transparente
simplicidade
não precisa de coordenação global
(não há ambigüidade entre processos)

1. *Request* para 243.0
2. *Replay* para 199.4



Implementação Cliente/Servidor
(Endereçamento)

- Endereçamento por processo
- Associar a cada processo um endereço único que não contém o número da máquina
- Duas alternativas para escolha do número do processo:

Transparência
Escalabilidade

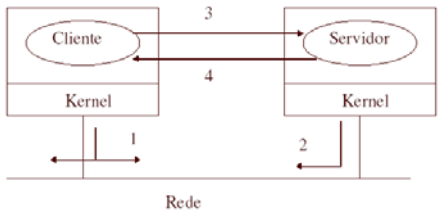
- Processo centralizado responsável pela alocação de endereços. (Contador incrementado a cada requisição)
- Cada processo pega seu próprio endereço aleatoriamente de um grande espaço de dados.

Transparência
Escalabilidade

Implementação Cliente/Servidor
(Endereçamento)

- o cliente pode fazer broadcast de um pacote especial para localização (locate packet)
  - o kernel que contém o processo devolve uma mensagem do tipo ***here I am***

Transparência
Gera carga extra no sistema



1. *Broadcast*
2. *Here I am*
3. *Request*
4. *Replay*

Implementação Cliente/Servidor
(Endereçamento)

- Endereçamento por name server
  - Utiliza uma máquina extra para mapear nomes de serviços em endereços de máquinas
  - servidores são referidos como strings e estas são embutidas nos programas

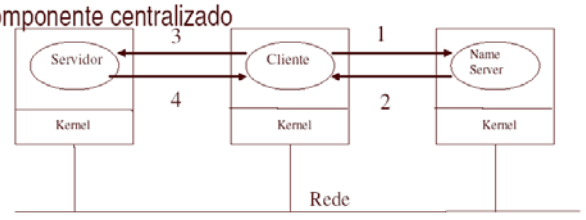
Implementação Cliente/Servidor
(Endereçamento)

- Endereçamento por name server

Transparente

Requer um componente centralizado

1. *Lookup*
2. *NS Reply*
3. *Request*
4. *Replay*



Implementação Cliente/Servidor

(Primitivas Bloqueantes ou Não-Bloqueantes)

- Primitivas Bloqueantes (síncronas)
  - no send, enquanto a mensagem está sendo enviada, o processo fica bloqueado
  - o receive fica bloqueado até que alguma mensagem chegue ou até um timeout
- Primitivas Não-Bloqueantes (assíncronas)
  - o send retorna o controle imediatamente, antes da mensagem ser realmente enviada
  - o receive passa para o kernel o ponteiro para o buffer e retorna imediatamente, antes de receber a mensagem
    - em algumas abordagens o receive não-bloqueante é aquele que só recebe quando já existem mensagens e fica bloqueado até completar a recepção

Implementação Cliente/Servidor

(Primitivas Bloqueantes)

- processo fica bloqueado durante a transferência de mensagem
- Melhor opção para envio de mensagens em condições normais
  - Simples de entender
  - Simples de implementar
  - Performance para envio de mensagem
  - CPU fica ociosa durante a transmissão

Implementação Cliente/Servidor

(Primitivas Não-Bloqueantes)

- Primitivas não-bloqueantes com cópia
  - o kernel copia a mensagem para um buffer interno e então libera o processo para continuar
- Primitivas não-bloqueantes com interrupção
  - interrompe o processo que enviou a mensagem quando o buffer estiver livre para reutilização

Implementação Cliente/Servidor

(Primitivas Não-Bloqueantes)

- Tanenbaum
  - A diferença essencial entre uma primitiva síncrona e uma assíncrona é se o processo que envia a mensagem pode reutilizar o buffer imediatamente após o comando send
  - preferida por projetistas de sistemas operacionais
- Andrews
  - Uma primitiva síncrona é aquela em que o processo que envia fica bloqueado até que o receptor aceite a mensagem e mande um ack. Qualquer outra alternativa é considerada assíncrona
  - preferida por projetistas de linguagens de programação

Implementação Cliente/Servidor

Bufferização

- Primitivas não-bufferizadas
  - O buffer para armazenar a mensagem deve ser especificado pelo programador
  - Existem duas estratégias a serem empregadas no caso de um send do cliente, sem um receive do servidor:
    - discartar mensagens inesperadas
    - temporariamente manter mensagens inesperadas
- Primitivas bufferizadas
  - Existe um buffer para armazenar mensagens inesperadas (Kernel)
  - A primitiva de bufferização mais empregada define estruturas de dados chamadas mailbox

Implementação Cliente/Servidor

Confiabilidade

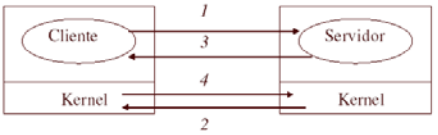
- Três diferentes alternativas podem ser utilizadas:
  - assumir que as primitivas não são confiáveis, alterando a semântica do send
    - o sistema não garante que as mensagens são enviadas
    - o usuário fica responsável por implementar comunicação confiável
  - primitivas confiáveis com mecanismos de acknowledgment do tipo:
    - Request - Ack - Reply - Ack
  - primitivas confiáveis com mecanismos de acknowledgment do tipo:
    - Request - Reply - Ack
  - combinações podem ser obtidas entre os mecanismos confiáveis

Implementação Cliente/Servidor

Confiabilidade

- Request - Ack - Reply - Ack
  - somente quando o Ack é recebido, o processo é liberado
  - o acknowledgement é feito entre kernels (transparente para o cliente ou servidor)
  - um request/reply com este mecanismo necessita de quatro mensagens

1. *Request*
2. *Ack*
3. *Reply*
4. *Ack*

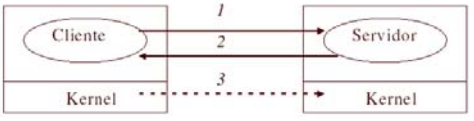


Implementação Cliente/Servidor

Confiabilidade

- Request - Reply - Ack
  - O Reply serve como um ack
    - o cliente fica bloqueado até a mensagen de reply
    - se a mensagem de reply demorar, o cliente reenvia a requisição
    - em alguns kernels não é necessário o ack

1. *Request*
2. *Reply*
3. *Ack*



Implementação Cliente/Servidor

Outras Questões

- As redes têm um tamanho máximo de pacote, mensagens maiores devem ser quebradas
- O acknowledgment pode ser utilizado por pacote ou por mensagem, dependendo da taxa de erros da rede

Implementação Cliente/Servidor

Exemplo de Protocolo

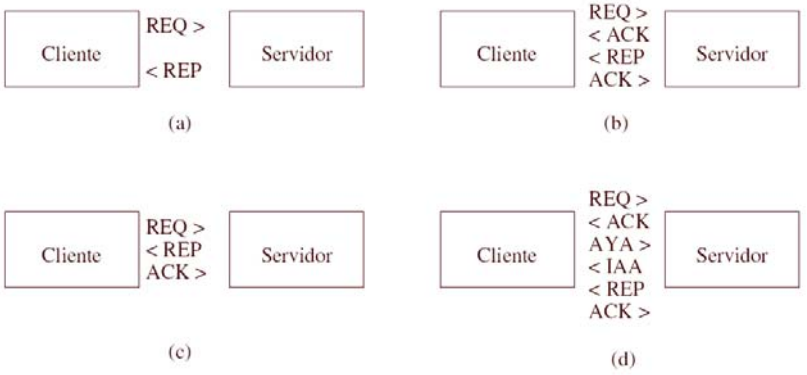
- Pacotes normalmente empregados no protocolo de comunicação:

REQ	Request	Cliente	Servidor	O cliente quer serviço
REP	Reply	Servidor	Cliente	Resposta do servidor para cliente
ACK	Ack	Cli./Ser.	Outro	O pacote anterior chegou
AYA	Are you alive?	Cliente	Servidor	Verifica se o servidor está Ok
IAA	I am alive	Servidor	Cliente	O servidor está Ok
TA	Try Again	Servidor	Cliente	O servidor não tem espaço
AU	Addr. Unknown	Servidor	Cliente	Nenhum processo usa o endereço

Implementação Cliente/Servidor

Comunicação usando o protocolo

- Alguns exemplos de comunicação:



Troca de Mensagens

- Envio de dados e controle pela rede para um ou mais participantes.
- Forma mais primitiva e comum, próxima à rede
- Mensagem é uma estrutura de dados
- Primitivas básicas são usadas aos pares:
  - envio: send(msg) ou send(destino, msg)
  - recepção: recv(&msg) ou recv(origem, &msg)
  - identificação de destino (endereçamento)? processo, CP, IP/porta...

Troca de Mensagens

- Serviço de envio:
  - processo P solicita envio de mensagem à Q
  - processo Q solicita recebimento de mensagem de P,ou um procedimento em Q é executado quando chega a mensagem

Troca de Mensagens

- Transmissão síncrona x assíncrona: quando o remetente é desbloqueado?
  - após a mensagem ter sido processada pelo receptor
  - após a mensagem ter sido entregue ao receptor
  - após a mensagem ter chegado ao nó receptor
  - após a mensagem ter partido do nó transmissor
  - após a mensagem ter sido copiada para os buffers do nó transmissor imediatamente

Troca de Mensagens

- Transmissão confiável x não confiável: que garantias?
- ok apenas se não houver nenhum tipo de falha (omissão,atraso...)
- ok se houver perdas de mensagens e se o remetente ou receptor(es) morrerem?

Troca de Mensagens

- Transmissão com x sem conexão
  - com conexão, remetente e receptor conversam para estabelecer uma conexão entre dois pontos (“endpoints”)
  - o que representa exatamente uma conexão? informações de estado em ambas as partes
  - o que é “stateless”?

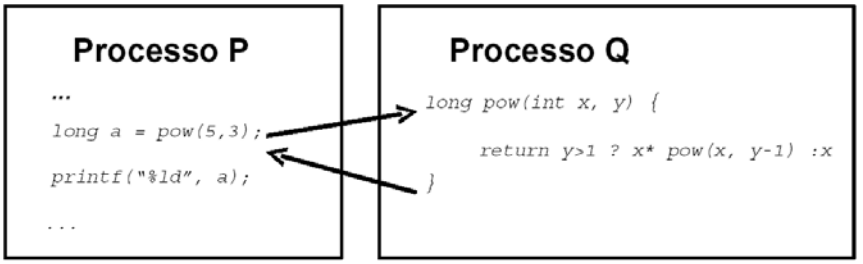
Troca de Mensagens

- Modelos (Os mecanismos tradicionais de transmissão de pacotes são) :
  - unicast
  - broadcast
  - multicast
  - anycast
  - concast

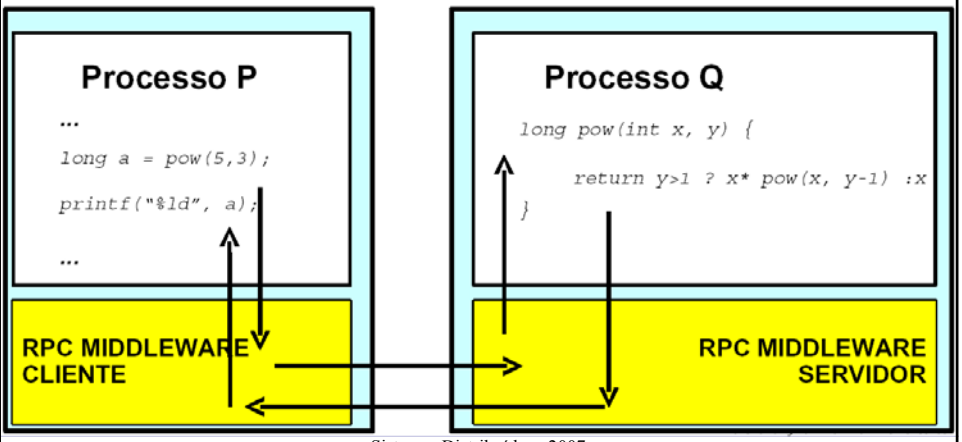
Chamada Remota de Procedimento (RPC)

- Desviando o fluxo de execução para uma máquina remota, passando argumentos e recebendo valores de resposta.
- Permite a um processo executar uma "subrotina" em um outro processo, possivelmente remoto
- Por exemplo, processo P executa função pow() que faz parte do Processo Q

Chamada Remota de Procedimento (RPC)



Chamada Remota de Procedimento (RPC)



Chamada Remota de Procedimento (RPC)

- Justificativa para criação de RPC foi que "passagem de mensagens" era um mecanismo complexo e dificultava desenvolvimento de aplicações distribuídas
- RPC "esconde" troca de mensagens em chamadas de procedimentos sintaxe próxima a chamadas em linguagens tradicionais facilitou conversão de aplicações legadas em distribuídas

Chamada Remota de Procedimento (RPC)

- Segue modelo cliente/servidor, em geral com interações síncronas
- Questão: mas como poderiam ser assíncronas?

Chamada Remota de Procedimento (RPC)

- Lado cliente:
  - aplicativo, lado cliente, que solicita serviço
  - "stub cliente", gerado automaticamente
  - "RPC runtime" do lado do cliente
- Lado servidor:
  - aplicativo, lado servidor, que recebe solicitação e processa
  - "stub servidor", gerado automaticamente
  - "RPC runtime" do lado do servidor
- Código do stub cliente e servidor são compilados

Chamada Remota de Procedimento (RPC)

- Problemas e Limitações:
  - Obter com RPC mesma semântica de chamada local é difícil, por diversas razões, conforme explicado a seguir...
  - Necessário fase de binding (amarração):
    - stub precisa primeiro localizar função (máquina e porta associadas)
    - uma solução é uma base de dados com localização das subrotinas, base de dados possui endereço fixo e conhecido

Chamada Remota de Procedimento (RPC)

- Problemas e Limitações:
  - Implementação de passagem de parâmetros por referência:
    - na ausência de memória compartilhada, apontadores não tem significado no processador remoto
  - Deve tratar exceções (deve estar indicado no código):
    - problemas na rede, morte processo servidor, morte do cliente durante a execução de chamada remota...

Chamada Remota de Procedimento (RPC)

- Problemas e Limitações:
  - Semântica das chamadas:
    - em chamada local, função é executada uma única vez
  - na rede, há perdas de mensagens e retransmissões, e falhas de hosts
  - há três semânticas diferentes para chamadas remotas: exatamente-uma-vez (exactly-once), no-máximo-uma (at-mostonce) , no-mínimo-uma (at-least-once)

Chamada Remota de Procedimento (RPC)

- Problemas e Limitações:
  - Heterogeneidade e representação de dados: arquiteturas possivelmente incompatíveis
    - conversão de dados entre diferentes representações
    - exemplos de incompatibilidades: ordem, precisão, código de caracteres
  - Desempenho: overhead é substancial e diminui desempenho por fator de 10+ em relação a mensagens
  - Segurança: permitir execução de procedimentos localmente pode criar "furos" da segurança

Comunicação Grupal

- Processos são organizados em grupos distintos e se comunicam através do envio de mensagens para os grupos, sendo as mesmas entregues de forma confiável e ordenada aos membros de um grupo.

Comunicação Grupal

- Aplicações cooperativas permitem a interação entre diversos usuários (implementado através da troca de mensagens)
- Além de transporte eficiente via multicast, estas aplicações demandam informação atualizada sobre a composição atual do grupo, ou *group membership*
- Para a aplicação, grupos podem ser
  - visíveis: p.ex., funcionalidade da aplicação mapeada em grupos
  - invisíveis: p.ex., réplicas de dados ou processamento
- Dois serviços:
  - serviço de composição de grupo
  - serviço de comunicação em grupo

Comunicação Grupal

Serviço de Composição de Grupo

- Serviço oferece duas funções aos participantes:
  - habilidade de criar grupos, entrar e sair de grupos
  - informações atualizadas sobre alcançabilidade mútua ("mutual reachability"), conhecido como visão de grupo (group view)
- Criar grupos, entrar em grupos, deixar grupos:
  - Join() e Leave()

Comunicação Grupal

Serviço de Composição de Grupo

- Determinação de alcançabilidade de outros membros realizada por detectores de defeitos (failure detectors)
- Mudanças devem ocasionar a entrega de uma nova visão a todos os membros do grupo, promovendo concordância entre os mesmos (sobre estado)

Comunicação Grupal

Serviço de Composição de Grupo

- Serviços variam em relação às garantias oferecidas:
  - ordem de entrega das visões aos membros
  - ordem de entrega das mensagens em relação às mudanças nas visões
- Um serviço de composição de grupo deveria prover duas propriedades fundamentais:
  - **precisão** (accuracy): a informação oferecida reflete o cenário físico
  - **consistência** (consistency): a informação oferecida é consiste para todos os processos
- ...apesar da ocorrência de falhas, o que é bastante difícil

Comunicação Grupal

Serviço de Comunicação em Grupo

- Serviço para permitir a transmissão multicast para todos os membros de um grupo, ou um subconjunto
- Dois aspectos principais:
  - **confiabilidade** (reliability): garantias de entrega de mensagens
  - **ordenamento** (ordering): garantias de ordenamento de mensagens
- Grupos fechados x grupos abertos

Comunicação Grupal

Serviço de Comunicação em Grupo

- Atomicidade:
  - necessário que todas os destinos recebam as mesmas mensagens
  - multicast atômico: ou todos receptores recebem, ou nenhum recebe
- Esquemas de ordenação:
  - sem ordenação
  - FIFO
  - ordenação causal
  - ordenação total
  - ordenação síncrona

Objetos Distribuídos

- Principais tecnologias
  - Java RMI
  - CORBA
  - COM/DCOM
- Vamos trabalhar no curso de SD com Java RMI
- Mais tarde vamos analisar e comparar o Java RMI com as demais tecnologias

Comunicação em SDs

- Outros mecasnimos
  - Message-oriented Middleware Systems (MOMS)
  - Memória compartilhada distribuída (DSM)
  - Web Services