Sistemas Distribuídos

Prof. Marcelo de Paiva Guimarães

Objetivos

- Apresentar uma visão geral de processamento distribuído, analisando os tópicos mais importantes sobre sistemas operacionais distribuídos, incluindo comunicação, sincronização, processos e sistema de arquivo.
- Discutir os problemas e abordagens nos projetos e desenvolvimento de Sistemas Distribuídos

Introdução

- De 1945 até 1985 temos computadores caros e grandes
- De 1985 em diante, o avanço da tecnologia mudou esta situação em dois pontos
  - Desenvolvimento de poderosos microcomputadores
  - Desenvolvimento de redes de computadores de alta velociadade
    - LANS (Local Area Networks) permitindo que até centenas de computadores dentro de um edifício sejam conectados, permitindo a troca de informações
    - WANs(Wide Area Networks) permitindo que milhões de computadores no mundo sejam conectados.

O que é um Sistema Distribuído ?

- Um SD é uma coleção de computadores independentes que aparenta aos usuários do sistema como se fosse um único computador
- A definição tem dois aspectos
  - Hardware: as máquinas são autônomas
  - Software: os usuários vêem o sistema como uma única máquina

Exemplos

- Redes de estações de trabalho em uma universidade
  - Workstation do usuário
  - Pool de processsadores
  - Único sistema de arquivos
- O sistema procura o melhor lugar para executar um determinado comando (própria estação;estação que não está sendo usado; pool de processadores)

Introdução aos Sistemas Distribuídos

- Sistemas Centralizados
  - única CPU
  - única memória
  - periféricos
- Sistemas Distribuídos
  - múltiplas CPUs
  - interconectadas de alguma forma
  - cooperando de alguma maneira

Sistemas Distribuídos

- Por que Sistemas Distribuídos ?
- Por que não um Sistema Centralizado com maior capacidade?
- Existem sistemas de natureza claramente distribuída
  - sistemas de produção de uma fábrica
  - sistema de reserva de passagem aérea
  - sistemas de automação bancária
  - Sistema de suporte ao trabalho cooperativo

Conceitos Diferentes ?

- Sistemas Distribuídos
  - vários usuários trabalhando em conjunto

- Sistemas Paralelos
  - um único problema resolvido mais rapidamente

Sistemas Distribuídos: Vantagens

- Solução mais efetiva em termos de custo : uso de um grande número de processadores baratos
- Possibilidade de conseguir um desempenho que seria impossível usando processadores centralizados
- Atender a novos requisitos de usuários
- Mais Vantagens:
  - maior confiabilidade em relação aos centralizados
  - crescimento incremental
  - expansão gradual

Vantagens sobre PCs isolados

- Já que os PCs ficaram tão bratos porque não dar um para cada usuário trabalhar de forma independente ? Não, devido:
  - À necesidade de se compartilhar dados
  - À necessidade de compartilhar periféricos caros
  - Á necessidade de comunicação pessoa a pessoa
  - À flexibilidade de poder espalhar o trabalho existente em diversas máquinas, balanceando a carga de trabalho

Desvantagens dos Sistemas Distribuídos

- Hardware é maduro, o Software ainda está evoluindo.
- Poucos softwares disponíveis
- Falta experiência no projeto, implementação e uso de software distribuído
- Quanto deve o sistema fazer e quanto deve o usuário fazer ?
- Redes de interconexão
  - perda de mensagens
  - Sobrecarga na rede
- Segurança

Conceitos de Software

- A imagem que o sistema apresenta ao usuário e o modo como eles pensam sobre o sistema é determinado principalmente pelo software
- S.O para sistemas com múltiplas CPUs
  - Fracamente Acoplados
  - Fortemente Acoplados

Conceitos de Software

- Software Fortemente Acoplado – os diversos processadores do sistema cooperam na execução das tarefas
  - Ex: Processamento de imagens
- Software Fracamente Acoplado – permite que as máquinas e usuários do sistema sejam independentes uns dos outros, interagindo em um grau limitado quando for necessário
  - Ex: Um grupo de computadores pessoais, cada um com a sua própria CPU, memória, disco rígido e S.O, compartilhando alguns recursos através de uma rede

Sistemas Distribuídos

- Fracamente Acoplado
  - independentes
  - cooperam de alguma forma
  - a falha de uma CPU acarreta pouca degradação ao sistema
- Fortemente Acoplado
  - divide um problema por várias CPUs
  - cada CPU trabalha em paralelo
  - um único resultado

Tipos de Sistemas

- Sistema Operacional de Redes
  - Software fracamente acoplado em hardware fracamente acoplado
- Sistemas Distribuídos (Multicomputador)
  - Software fortemente acoplado em hardware fracamente acoplado
- Sistemas Timesharing Multiprocessadores
  - Software Fortemente Acoplado em Hardware Fortemente Acoplado

Sistema Operacional de Redes

- Software fracamente acoplado em hardware fracamente acoplado
  - Ex: Rede de estações de trabalho conectadas por uma LAN
  - Cada usuário tem a sua estação de trabalho
  - Cada estação de trabalho pode ter ou não disco rígido
  - Cada estação de trabalho tem o sue próprio sistema operacional
  - Todos os comandos são normalmente executados localmente
  - Eventualmente é possível fazer uma conexão remota com outra estação de trabalho

Sistema Operacional de Redes

- Forma mais conveniente de comunicação e compartilhamento de informação:
  - Sistema de Arquivo Global Compartilhado
    - Implementado em uma ou mais máquinas chamadas Servidores de Arquivos
    - Servidores de Arquivos aceitam requisições de programas de usuários

Sistemas Distribuídos

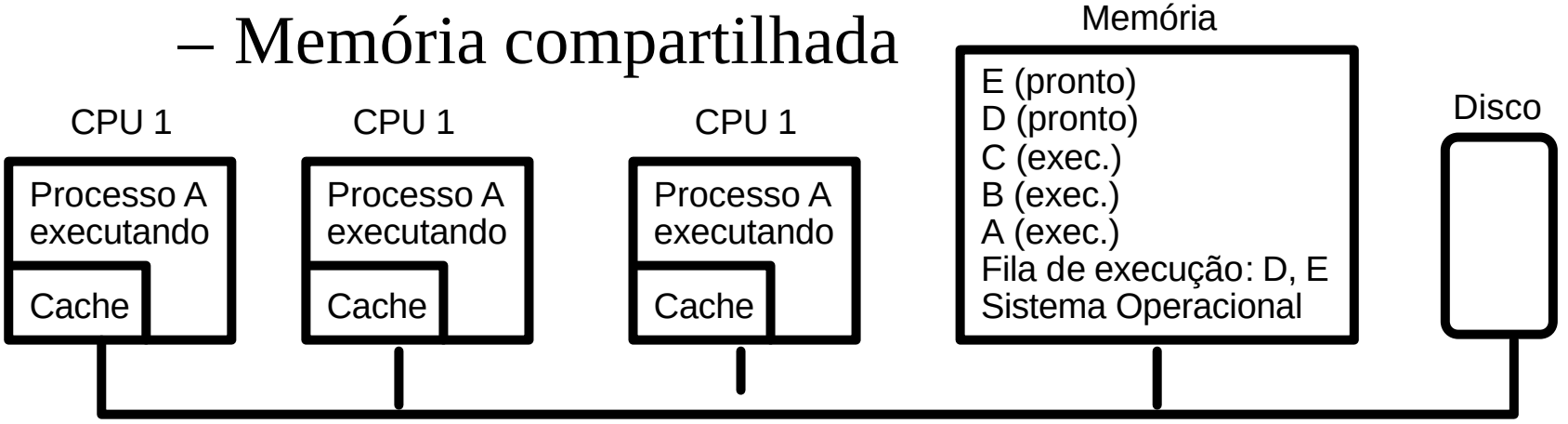
- Software fortemente acoplado em hardware fracamente acoplado (Multicomputador)
  - O objetivo é criar a ilusão para os usuários de que a rede de computadores é um único sistema timesharing

Sistemas Distribuídos Verdadeiros

- Características
  - single system image (imagem de um sistema único)
  - virtual uniprocessor
  - usuário não precisa saber da existência de várias CPUs
- Importante
  - mecanismo para Comunicação Interprocessos (IPC)
  - esquema global de proteção
  - gerenciamento de processo uniforme em todas as máquinas (criação, destruição, começo, interrupção)
- conseqüência
  - Cópias identicas do kernel executam em todas as CPUs do sistema (escalonamento, swapping, paginação, etc)

Sistemas Timesharing Multiprocessados

- Software Fortemente Acoplado em Hardware Fortemente Acoplado
  - Característica chave: existência de apenas um fila de pronto (run quee)
    - Exclusão mútua:
    - monitores
    - semáforos
  - Memória compartilhada



Diferença entre 3 tipos de sistemas

	SO Rede	SO Distrib.	SO Mult.
Ele se parece com um único uniporcessador virtual ?	NÃO	SIM	SIM
Tem o mesmo SO ?	NÃO	SIM	SIM
Número de cópias do SO	N	N	1
Como é a comunicação ?	Arquivo Compartilhado	Mensagens	Memória Compartilhada
Protocolos requeridos ?	SIM	SIM	NÃO
Uma única fila de execução ?	NÃO	NÃO	SIM
Arquivos tem a mesma semântica ?	Usualmente não	NÃO	SIM

Aspectos de Projeto dos Sistemas Distribuídos

- Transparência
  - Até onde e o quanto é desejável ?

- "Como é que o projetista de sistemas distribuídos *engana* a todos fazendo com que pareça ao usuário que a coleção de máquinas se comporte como o velho conhecido sistema de tempo compartilhado monoprocessado ?"

Transparência

- Pode ser conseguida em dois níveis diferentes
  - Esconder a distribuição do usuário.
    - make no unix
  - Em um nível mais baixo
    - A interface das chamadas do sistema pode ser projetada de tal forma que as existência de múltiplos processadores não seja visível.

Transparência

- Transparência de Localização
  - os usuários não precisam (devem ?) saber onde os recursos de hardware e software estão exatamente localizados.
  - serviço de nomes (diretório)
- Transparência para Migração
  - os recursos devem ser livres para mover de uma localização para outra sem a necessidade de alteração de nomes.
  - propriedade interessante para tolerância a falhas

Transparência

- Transparência para Replicação
  - o sistema operacional deve permitir a criação de novas cópias de recursos
  - o número de replicas deve ser transparente ao usuário
  - o estado das réplicas deve ser consistente
  - a criação de réplicas pode ser expontânea (Sistema Operacional) ou intencional (programado)
- Transparência para Concorrência
  - vários usuários compartilhando os mesmos recursos (sem relacionamento entre os mesmos)
  - sistema de lock automático para os recursos
- Transparência quanto ao Paralelismo
  - Atividades podem acontecer em paralelo sem o conhecimento dos usuários

Transparência

Localização	O usuário não pode dizer onde os recursos estão alocados
Migração	Recursos podem ser movidos sem troca de nomes
Replicação	Os usuários não sabem quantas cópias existem
Concorrência	Múltiplos usuários podem partilhar recursos automaticamente
Paralelismo	Atividades podem acontecer em paralelo sem o conhecimento dos usuários

Flexibilidade

- Permitir alterações e modificações no próprio sistema operacional
- Conceitos atuais podem não ser os melhores no futuro.
- Como a tecnologia ainda não está consolidada, a flexibilidade é importante como um meio para caminhos alternativos

Confiabilidade

- Teoria:
  - se uma máquina falhar, outra máquina continua o serviço, sem prejuízo do cliente
- Prática:
  - o sistema distribuído depende de um número de máquinas-chave

Lamport: "Sistema Distribuído é aquele onde você não consegue trabalhar porque alguma máquina, da qual você nunca ouviu falar, saiu do ar..."

Disponibilidade

- Disponibilidade
  - fração de tempo que o sistema é “usável”.
  - Disponiblidade pode ser aumentada por meio de redundância:
    - partes do hardware ou software podem ser replicados
  - Quanto maior o número de cópias de software, maior a probabilidade de inconsistências.
- Componentes críticos devem estar operando
- Não requer funcionamento simultâneo de componentes críticos
- Redundância (consistência de réplicas)

Segurança

- uso não autorizado
  - interrupção
  - interceptação
  - modificação
  - fabricação
- identificação de autoridade nas mensagens
- criptografia

Desempenho

- Um Sistema Distribuído não será útil se não tiver desempenho razoável
- Como medir desempenho ?
  - Tempo de resposta
  - Throughput (número de jobs por hora)
  - Taxa de utilização do sistema
  - Capacidade de rede consumida

Escalabilidade

- Escalabilidade
  - "soluções que funcionam para 200 máquinas podem falhar completamente para 200.000.000 máquinas"
  - algoritmos centralizados não são convenientes para Sistemas Distribuídos e devem ser evitados.
- Aspectos de algorítmos descentralizados:
  - nenhuma máquina tem informação completa do estado do sistema
  - máquinas devem tomar decisões apenas baseadas em informações locais disponíveis
  - a falha de uma máquina não deve comprometer o algoritmo
  - Não existe um relógio global(nem sincronismo entre relogios locais)

Comunicação

- Comunicação é relativamente lenta
- Granularidade das aplicações distribuídas
  - ex.: vale a pena a execução de uma soma remotamente ?
  - Paralelismo de granularidade fina
    - Grande número de pequenas computações, altamente interativas umas com as outras;
  - Paralelismo de granularidade grossa
    - Grandes computações, poucas interações
- Tolerância a Falhas tem seu preço em desempenho

Gargalos potenciais que devem ser evitados

Conceito	**Exemplo**
Componentes centralizados	Um único servidor de mail para todos os usuários
Tabelas centralizadas	Uma única lista telefonica on-line
Algorítmos Centralizados	Roteamento baseado em informação completa da rota

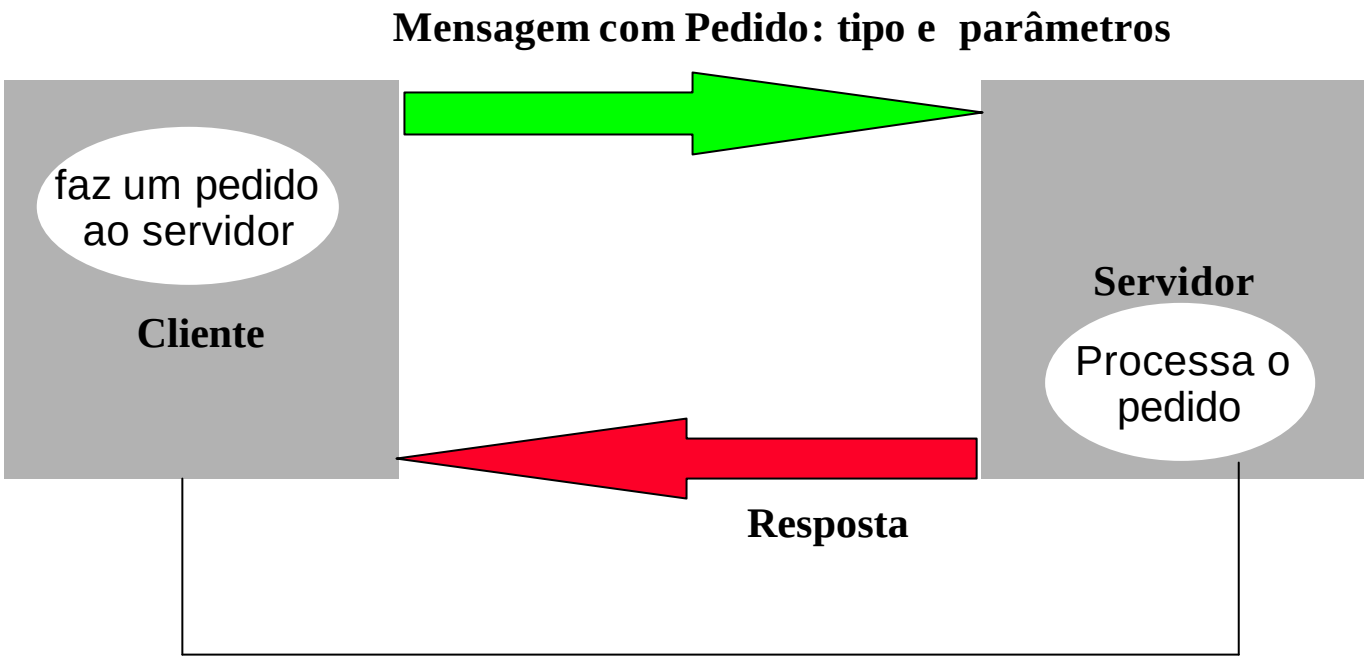
Comunicação

- A diferença mais importante entre os SD e os Sistemas Uniprocessadores é a comunicação intra processo.

- Nos uniprocessadores esta comunicação é feita por meio de memória compartilhada

- Nos SD não tem memória compartilhada.
  - Toda comunicação inter-processo deve ser reformulada

Modelo Cliente-Servidor



Modelo Cliente -Servidor

- A idéia é estruturar o S.O como um grupo de processos cooperativos chamados Clientes e Servidores
- Servidores oferecem serviços e Clientes usam os serviços oferecidos
- Uma máquina pode executar um único processo cliente ou servidor, pode executar múltiplos clientes, múltiplos servidores, ou ainda uma combinação dos dois

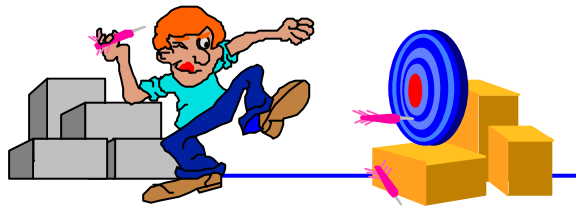
Principais vantagens do modelo Cliente-Servidor

- Simplicidade
  - Não tem necessidade de estabelecer uma conexão (e finalizar conexão)
- Eficiência
  - 3 camadas são necessárias (físico, data link e Request/Reply)
  - Apenas 2 primitivas básicas
    - send (dest, &mptr)
    - receive (addr, &mptr)

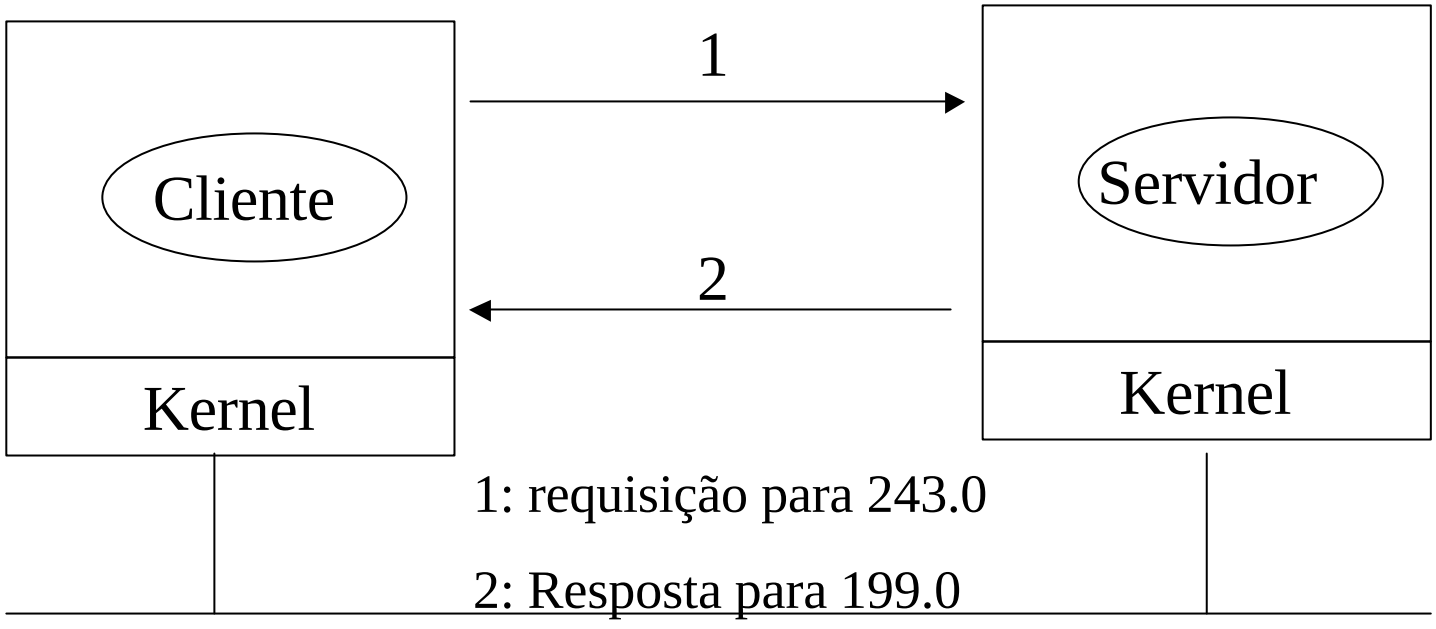
Interface

- Interface entre cliente/servidor
  - Bem definida
    - O cliente precisa saber como pedir
    - O servidor precisa saber como responder
  - Definida antes do diálogo
  - Formato da mensagem
  - exemplo: SQL

Endereçamento



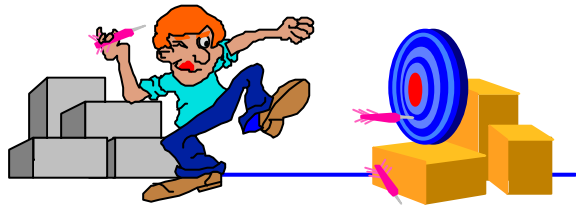
- Endereçamento **máquina.processo**



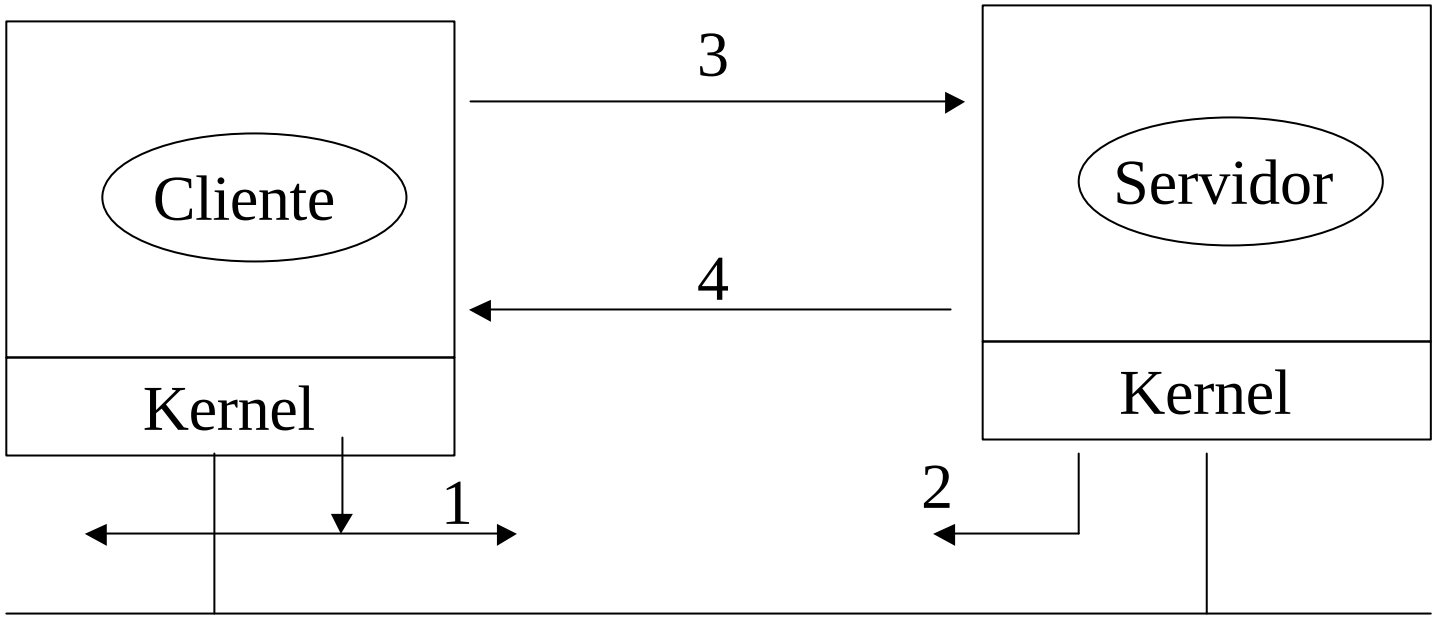
• Não é transparente (se um servidor não estiver disponível teremos recompilação para poder realizar o serviço em outro servidor

• Não precisa de coordenação global

Endereçamento

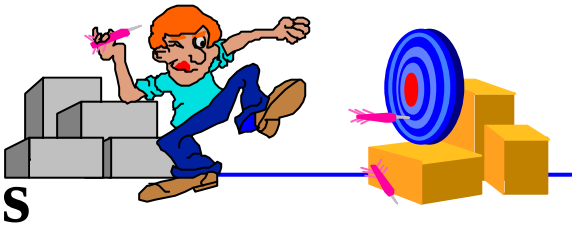


- Endereçamento **randômico**

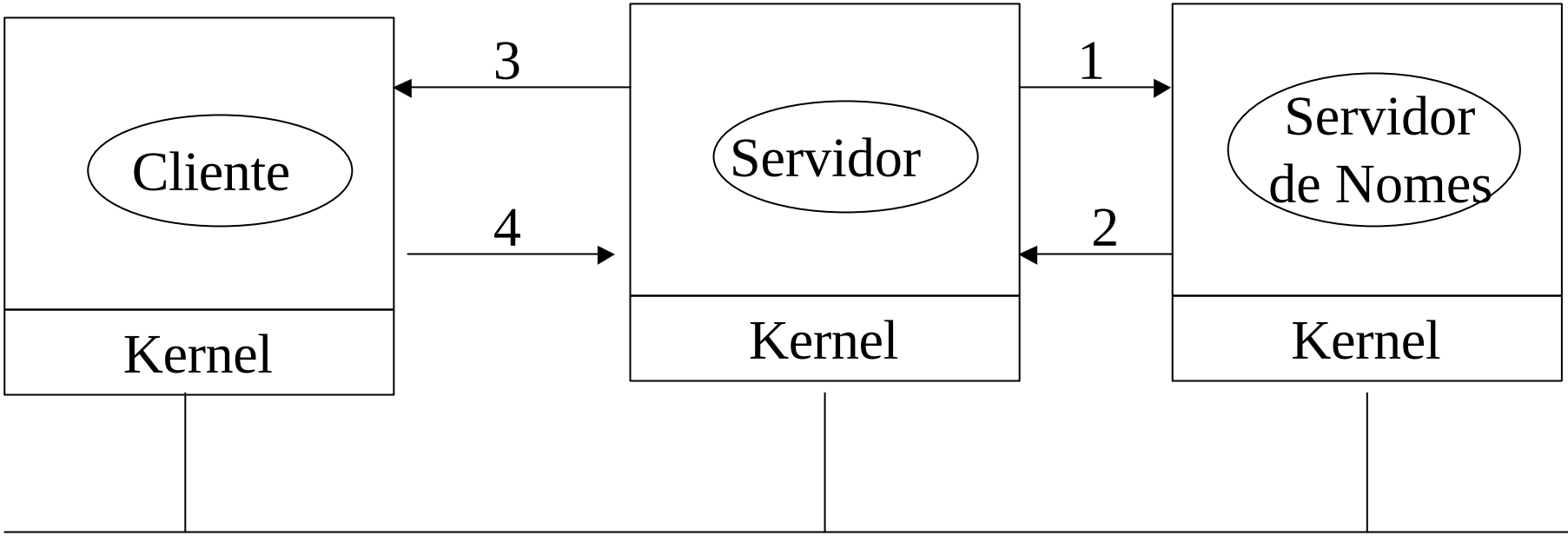


- 1: Broadcast
- 2: Estou aqui
- 3: Requisição
- 4: Resposta

Endereçamento



- Endereçamento **Servidor de Nomes**



- 1: Buscar nome
- 2: Resposta do Servidor de Nomes
- 3: Requisição
- 4: Resposta

Bloqueantes e Não-Bloqueantes

- Bloqueantes
  - Primitivas vistas até agora (send receive) são chamadas primitivas bloqueantes (síncronas)
  - Enquanto a mensagem está sendo enviada ou recebida, o processo permanece bloqueado (suspenso)
- Não bloqueantes
  - Assíncronas
  - Quando um send é executado o controle retorna ao processo antes da mensagem ser enviada
  - O processo que executa o send pode continar a executar enquanto a mensagem é enviada
  - A escolha do tipo de primitiva a ser usado (bloqueada ou não bloqueada) é feita pelo projetista do sisteam (normalmente é usado um dos dois tipos, em poucos caso, temos os dois tipos disponívies)

Bloqueantes e Não-Bloqueantes

- Não bloqueantes
  - O processo que executa o send não pode modificar o buffer de mensagem até a mensagem ter sido enviada
  - Modos para o processo saber que a mensagem já foi enviada
    - Send com cópia – desperdício do tempo de CPU com a cópia extra
    - Send com interrupção – programação muito mais difícil

Bloqueantes e Não-Bloqueantes

- Não bloqueantes
  - Primitiva receive não bloqueada
    - Simplesmente informa o kernel onde é o buffer para receber a mensagem e retorna o controle ao processo que o executou
    - Modos do receive saber que a recepção já acabou
      - Primitiva wait
      - Receive condicional
      - Interrupção

Bloqueantes e Não-Bloqueantes

- Timeout
  - Nas primitivas bloqueadas geralmente é usado um “timeout” para que as primitiva não fique bloqueada indefinidamente
  - É especificado um tempo de resposta depois do qual, se nenhuma mensagem chegar, o send termina retornando um código de erro

Primitiva Bufferizada e Não Bufferizada

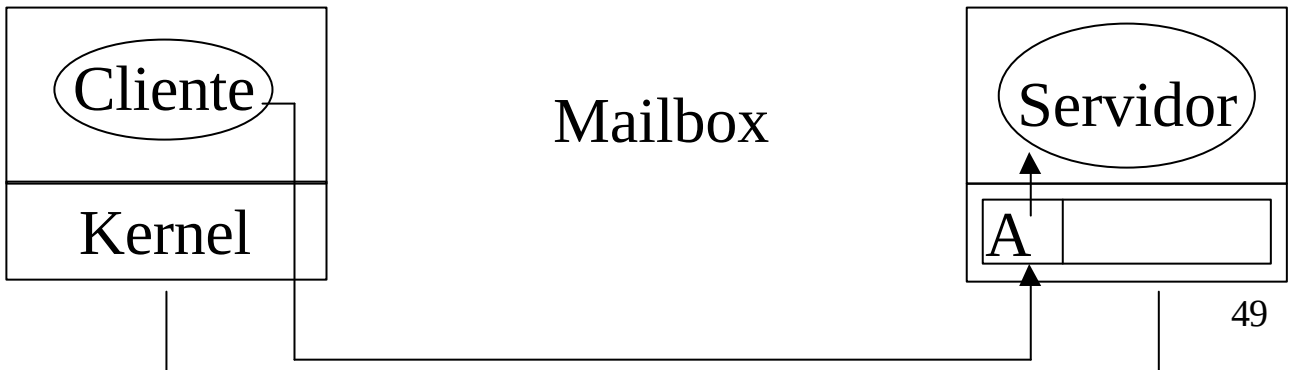
- As primitivas até agora são não-bufferizadas.
- O que acontece com um processo que chama receive quando o receive é executado depois do send ? (a mensagem chega antes do servidor chamar o receive)
- Como o kernel sabe onde colocar a mensagem que chega ? Não sabe !!

Duas soluções possíveis

- 1ª solução:
  - Descartar a mensagem e esperar o timeout expirar.
  - A mensagem é então retransmitida
  - Assim, a mensagem (“perfeita para ser consumida”) é descartada
  - O cliente pode assumir que o servidor falhou, erradamente
  - multiplos pedidos consecutivos -> alguns clientes não serão atendidos
- 2ª solução:
  - O kernel receptor "segura" as mensagens que chegam
  - Liga um timer
  - Reduz a chance da mensagem ser descartada
  - Problema: agora o kernel tem que gerenciar as msgs que chegam.
  - Solução: estrutura caixa postal. Processo, agora, se cadastra para ter uma caixa postal e remove a msg. da caixa postal.

Primitivas buferizadas

- Caixa Postal:
  - novo problema: caixa postal também é finita
  - descartar as msgs. ?
    - o problema é apenas amenizado.
  - Outra opção:
    - processo não envia msg. enquanto não houver espaço na caixa postal de destino
    - processo fica bloqueado até receber ACK
  - caixa postal cheia: processo é suspenso retroativamente, antes da primitiva send



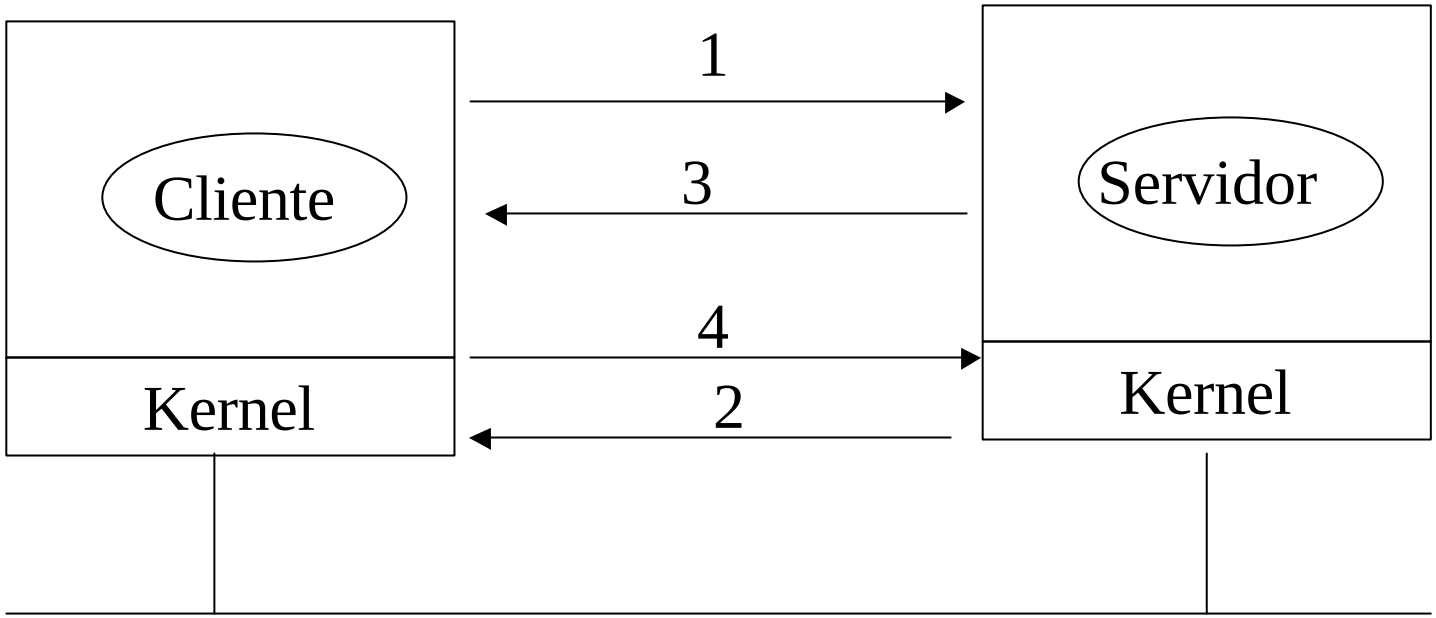
Primitivas Confiáveis e Não-Confiáveis

- problema principal:
  - mensagens podem se perder.
  - processo envia a mensagem e fica bloqueado até o núcleo despachar a mensagem
  - o núcleo despacha a mensagem e desbloqueia o processo
  - processo não tem garantia de que a mensagem despachada chegou ao destino

Soluções

- Solução 1 :
  - a primitiva send passa a ser não confiável
  - a confiabilidade é garantida pelo próprio usuário
  - será feito o melhor esforço para a mensagem chegar
- Solução 2:
  - o kernel da máquina que recebe a mensagem envia um ACK
  - o processo só é desbloqueado do send quando chega o ACK
  - o ACK é de kernel para kernel
  - um transação REQUEST/REPLY gasta 4 msgs.

Mensagens de ACK individual



1: requisição (Cliente para Servidor0

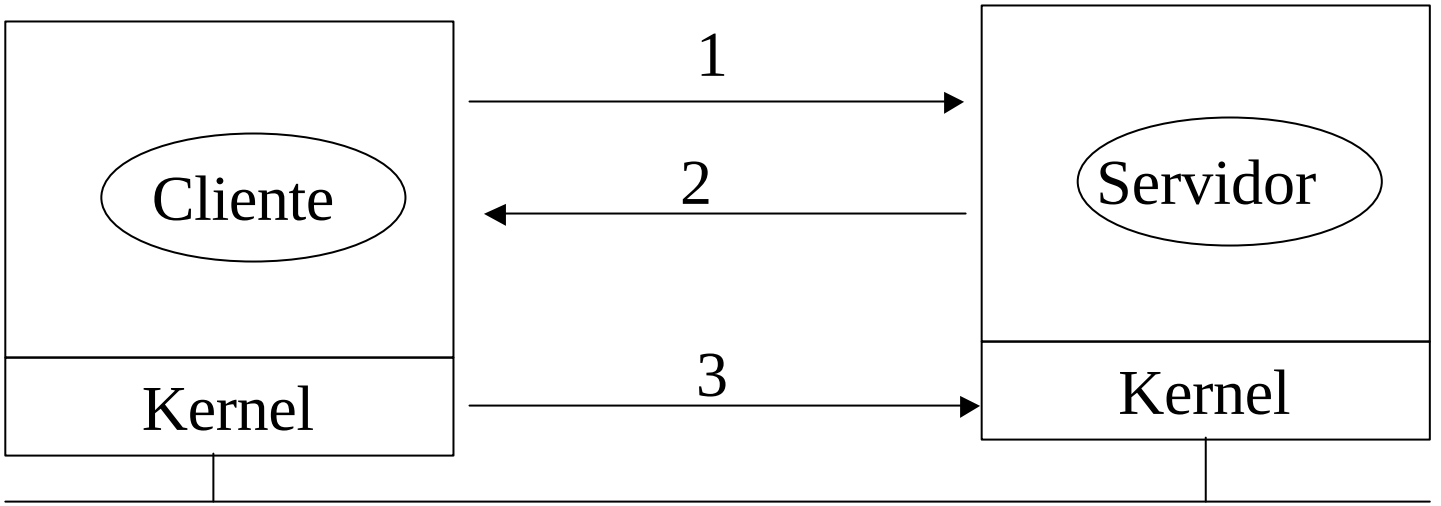
2: ACK (Kernel para Kernel)

3: Resposta (Servidor para Cliente)

4: ACK (Kernel para Kernel)

Resposta sendo usada como ACK

- Solução 3:
  - Aproveitar o fato que a comunicação cliente-serivdor é estruturada como uma requisição do cliente para o servidor seguido de uma resposta do servidor para o cliente



1: Requisição (Cliente para Servidor)

2:Resposta (Servidor para Cliente)

Implementação do Modelo Cliente-Servidor

Item	Opção 1	Opção 2	Opção 3
Endereçamento	Número de Máquina	Endereçametno Aleatório	Nomes ASCII com Servidor
Bloqueamento	Primitivas Bloqueadas	Não-Bloqueada com cópia para o kernel	Não-bloqueada com Interrupção
Buferização	Não-buferizado, decatando mensagens não esperdas	Não-buferizado, mantendo temporariamente mensagens não esperadas	Mailboxes
Confiabilidade	Não-confiável	Requisição-ACK – Resposta-ACK	Requisição – Resposta- ACK

Implementação do Modelo Cliente-Servidor

- Detalhes de como a passagem de mensagem é implementada depende das escolhas feitas durante o projeto. Algumas considerações
  - Há um tamanho máximo do pacote transmitido pela rede de comunicação
  - Mensagens maiores precisam ser dividdas em múltiplos pacotes que são enviados separadametne
  - Alguns dos pacores podem ser perdidos ou chegar na ordem errada.
    - Solução: atribuir a cada mensagem o número da mensagem e um número de sequência

Implementação do Modelo Cliente-Servidor

- O “acknowledgement” pode ser :
  - Para cada pacote individual
    - Na perda de mensagem, somente um pacote precisa ser retransmitod, mas na situação normal requer mais pacotes na rede de comunicação
  - Para a mensagem como um todo
    - Na perda de mensagem, há a vantagem de menos pacotes na rede mas a desvantagem da recuperação no caso de perda de mensagem é mais complicada
- Conclusão: a escolha de um dos dois métos depende da taxa de perdas na rede