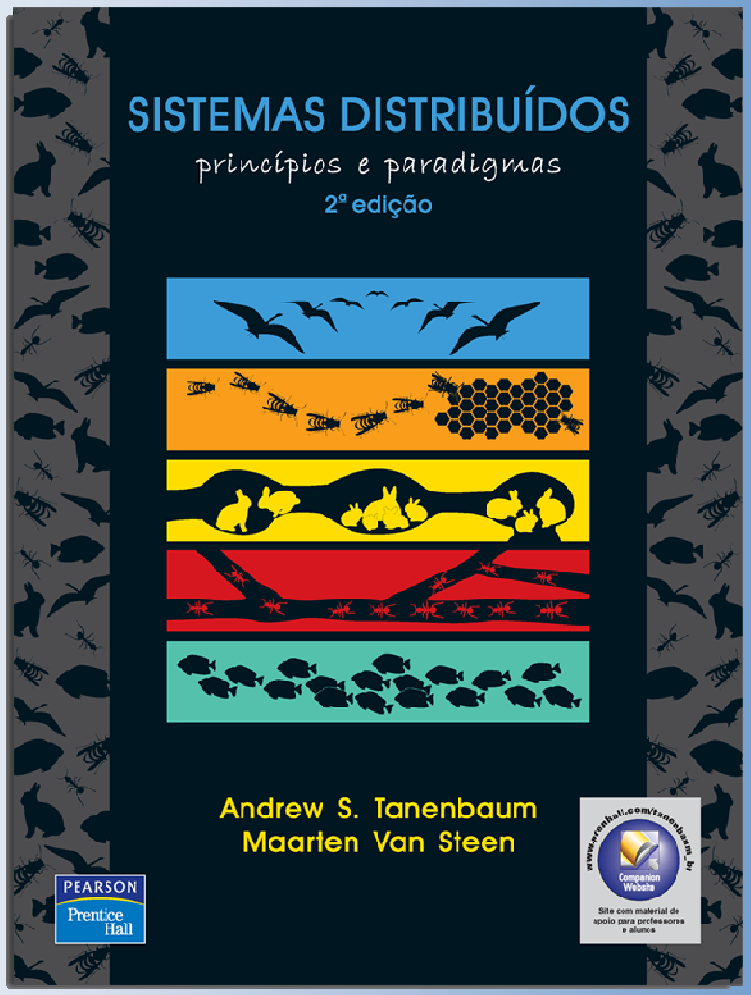
Processos



capítulo 3

Processos

- Diferentes tipos de processos desempenham papel crucial em sistemas distribuídos.
- O conceito, que vem de S.O., define processos como programas em execução.
- Usar processos multithreading auxiliam em melhoria de desempenho em sistemas cliente-servidor

Threads

- Vários *processadores virtuais* criados pelo S.O. são monitorados auxiliados pelo uso de uma *tabela de processos* que contém entradas para armazenar valores de registro de CPU, mapas de memória, entre outros.
- Um processo em execução está sendo executado em um dos processadores virtuais do S.O. no momento.

Threads

- O S.O. garante transparência sobre uso de CPU e dispositivos de I/O concorrentemente. Para isso é usado o escalonamento de processos.
  - Uso demasiado de chaveamento no modo dual (usuário/monitor);
  - Uso demasiado de chaveamento entre processos;
  - Modificações constantes no MMU (Memory Management Unit) e no TLB (Translation Lookaside Buffer)

Threads
Uso em sistemas não distribuídos

- Distribuição de várias tarefas concorrentes, sem que o processo inteiro seja bloqueado em espera a determinada resposta;
- Em sistemas multi-core, cada thread pode ser executada ao mesmo tempo em processadores distintos;
- Cooperação entre programas através do uso de IPC (InterProcess Communication)
  - Comunicação requer chaveamento de contexto em 3 pontos diferentes.

Threads
Chaveamento de Contexto - IPC

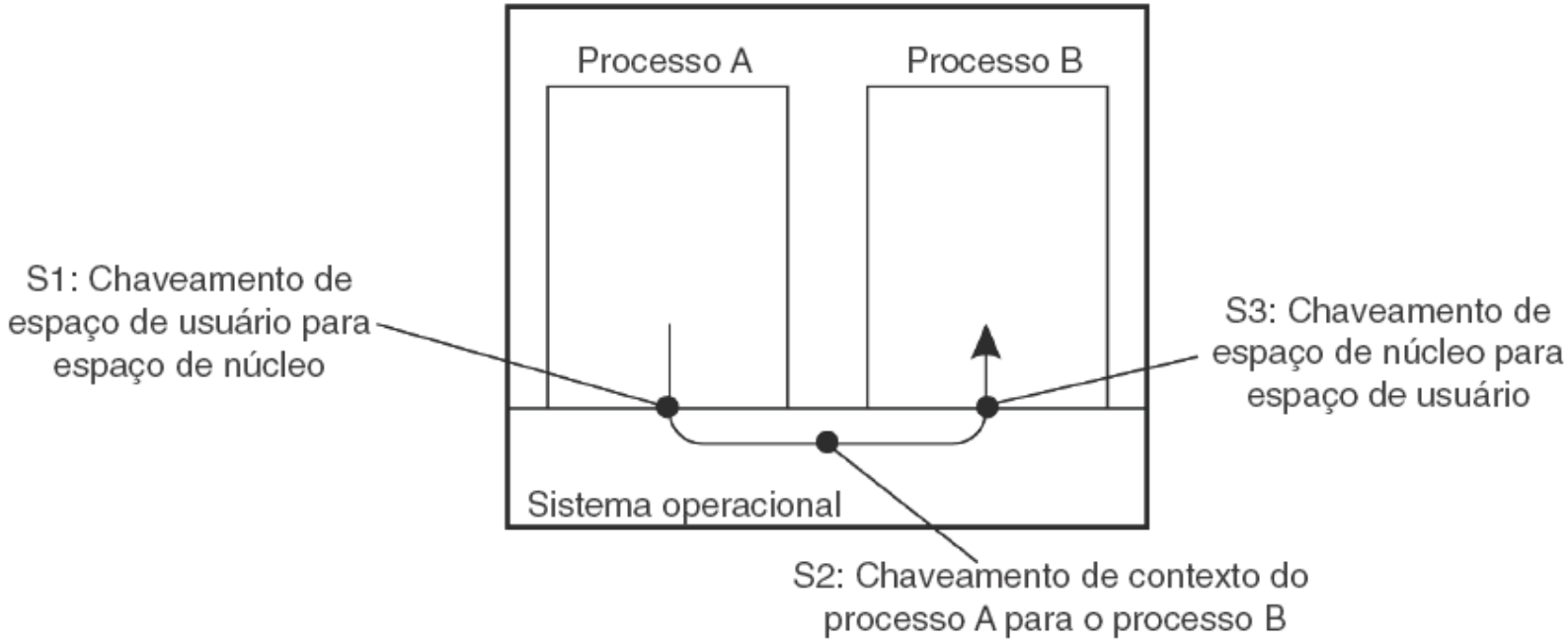


Figura 3.1 Chaveamento de contexto como resultado de IPC.

Implementação de Thread

Existem basicamente duas abordagens:

- Construir uma biblioteca de threads que é executada inteiramente em modo usuário:
  - Criar e terminar threads é barato;
  - Escalonamento é feito internamente;
  - Uma chamada bloqueadora bloqueia todo o processo.
- Fazer com que o núcleo fique ciente dos threads e os escalone
  - Criar e terminar threads tem alto custo;
  - Escalonamento feito pelo S.O.;
  - Chamada bloqueadora bloqueia apenas o thread

Implementação de Thread
LWPs

- Abordagem híbrida: LWP (Lightweight Process)
  - Executa em um único contexto (pesado);
  - Vários LWPs por processo;
  - Todas operações em threads são realizadas sem intervenção do núcleo;
  - Sincronização entre LWPs não requer intervenção do núcleo;
  - Uma chamada bloqueante bloqueia um LWP, mas não os outros LWPs, que compartilham a tabela de threads entre si.

Uso de LWPs para implementação de threads

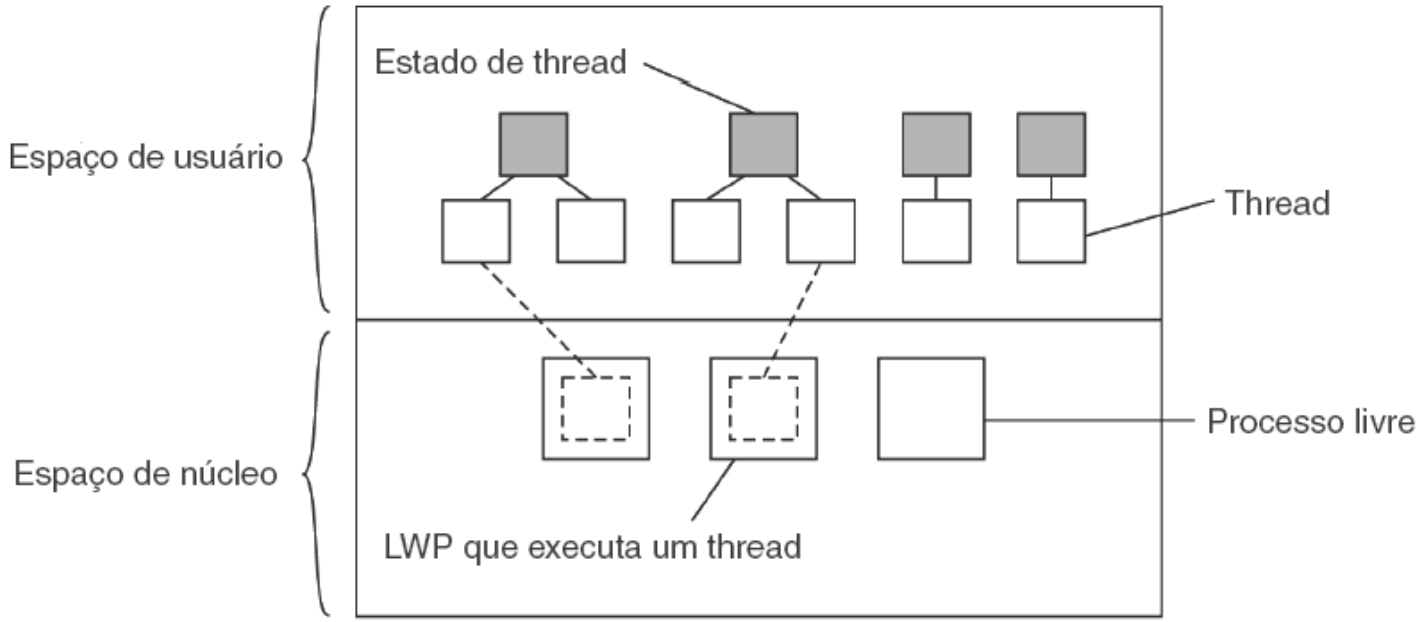


Figura 3.2 Combinação de processos leves de nível de núcleo e threads de nível de usuário.

Threads em Sistemas Distribuídos

- Proporcionam um meio conveniente para permitir chamadas bloqueadoras de sistema sem bloquear o processo inteiro no qual o thread está executando;
- Imagine um processo monothread: O que aconteceria com o processo quando a interrupção da placa de rede é feita para envio/recebimento de dados?

Clientes Multithread

- Usados para ocultar latências de comunicação, separando threads de envio/recebimento de dados com threads de processamento da interface.
  - Torna possível recebimento de vários arquivos de uma página WEB ao mesmo tempo;
  - Torna possível acesso a vários servidores (redundantes), que servirão os dados independentemente, gerando maior velocidade.

Servidores Multithread

- Além de simplificar o código do servidor, explora paralelismo para obter alto desempenho, mesmo em sistemas monoprocessadores;
  - Um thread despachante cria a divisão de vários threads com tarefas distintas, como ler disco, receber dados de socket, enviar dados para socket, atender N usuários simultaneamente;
  - O thread despachante atribui a requisição a um thread operário ocioso (bloqueado).
- Servidores Monothread não poderiam atender a um segundo usuário enquanto lê disco!

Servidores Multithread
Modelo despachante/operário

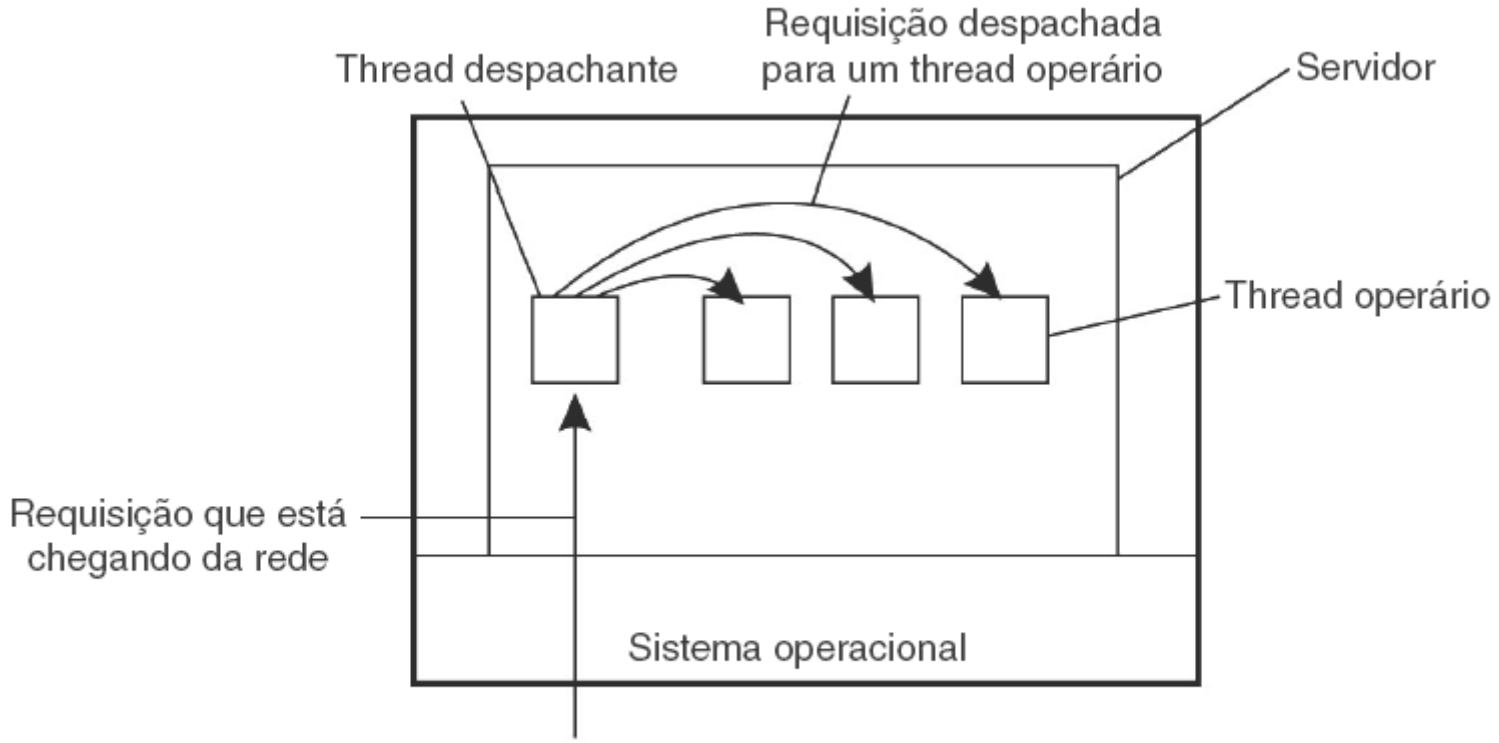


Figura 3.3 Servidor multithread organizado segundo modelo despachante/operário.

Servidores Multithread

- Uma terceira alternativa seria usar uma máquina de estados finitos, que consiste em apenas um thread que usa chamadas não bloqueadoras como *send* e *receive*.

Modelo	Características
Threads	Paralelismo, chamadas bloqueadoras de sistema
Processo monothread	Sem paralelismo, chamadas bloqueadoras de sistema
Máquina de estado finito	Paralelismo, chamadas de sistema não bloqueadoras

Tabela 3.1 Três modos de construir um servidor.

Virtualização e seu papel em S.D.s

- É a capacidade de uma única CPU em fingir que há mais delas, assim como essa extensão a outros recursos.
  - Estende ou substitui uma interface existente para imitar o comportamento de outro sistema

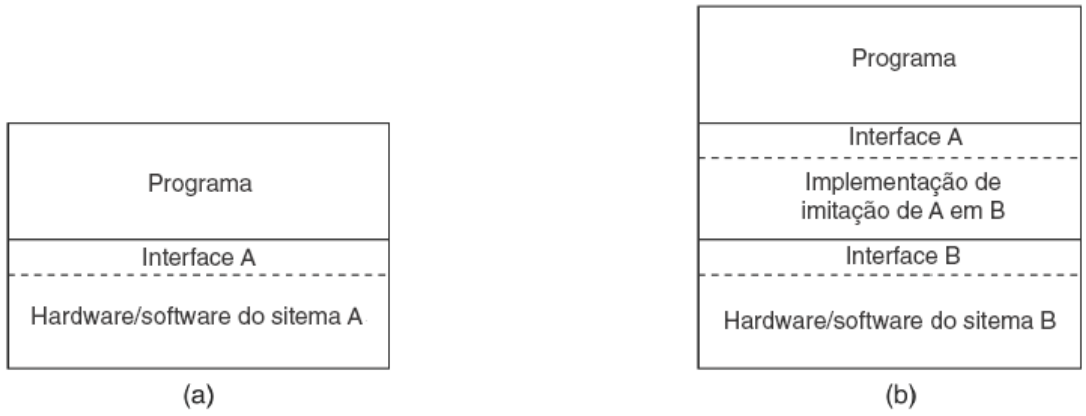


Figura 3.4 (a) Organização geral entre programa, interface e sistema. (b) Organização geral da virtualização do sistema A sobre o sistema B.

Virtualização
Arquiteturas de máquinas virtuais

- Sistemas de computadores oferecem quatro tipos diferentes de interface em quatro 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
Interfaces

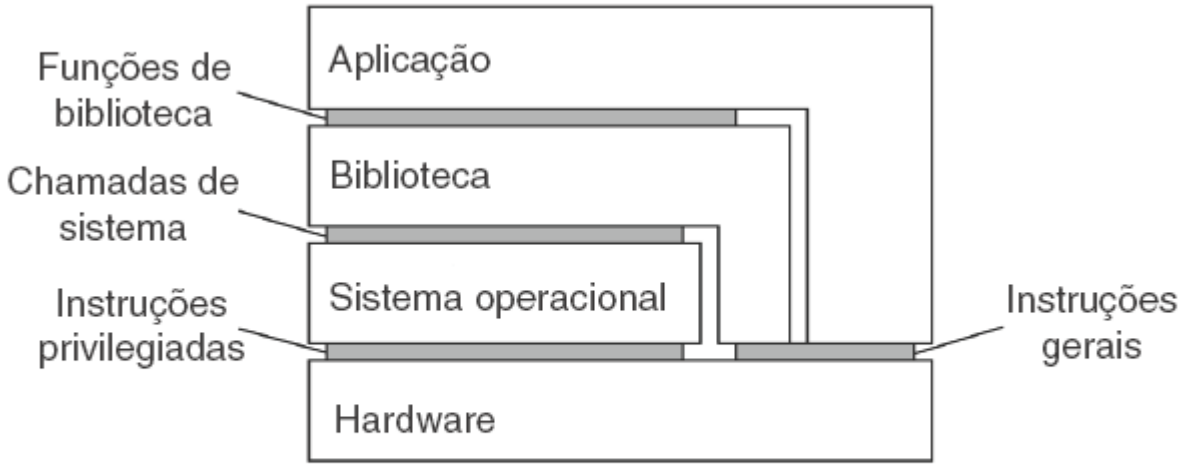


Figura 3.5 Várias interfaces oferecidas por sistemas de computadores.

Virtualização
Arquiteturas de máquinas virtuais

- A virtualização pode ocorrer de dois modos:
  - Construir um sistema de execução que forneça um conjunto abstrato de instruções que deve ser usado para executar aplicações – **máquina virtual de processo** (interpretadas, como Java);
  - Fornecer um **monitor de máquina virtual**, que 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.

Arquiteturas de máquinas virtuais
PVM x VMM

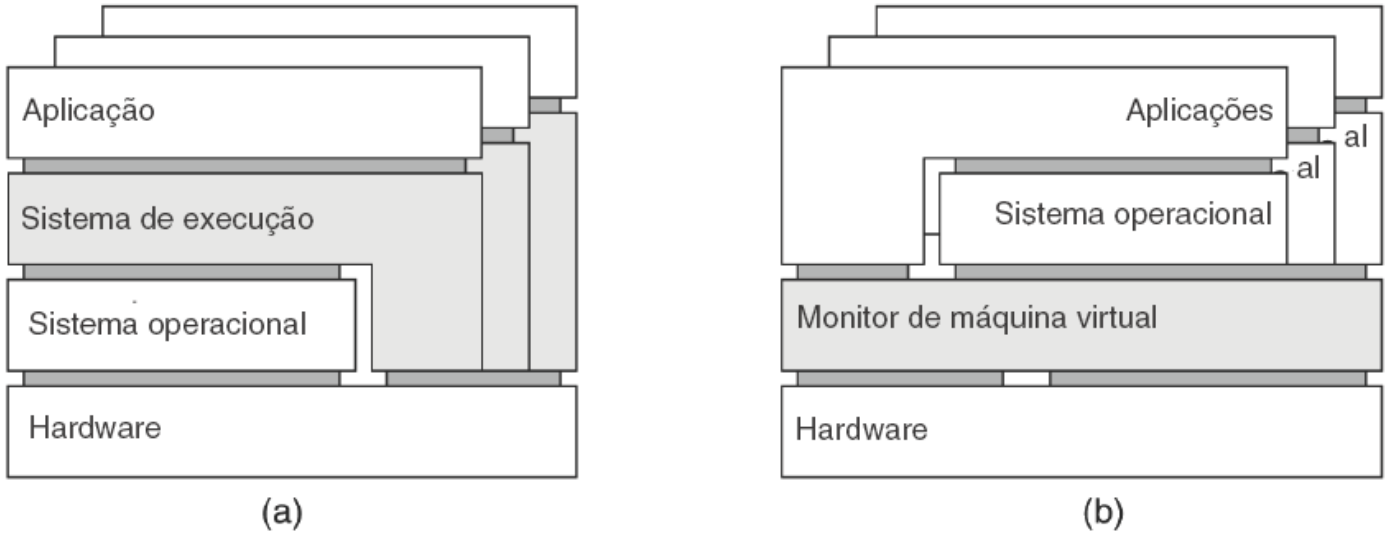


Figura 3.6 (a) Máquina virtual de processo, com várias instâncias de combinações (aplicação, execução). (b) Monitor de máquina virtual com várias instâncias de combinações (aplicações, sistema operacional).

Clientes

- Proporcionam meios para interagir com servidores, podendo ser das seguintes formas:
  - Sincronizados por protocolo de aplicação;
  - Através de serviços remotos, oferecendo apenas uma interface de usuário conveniente.

Clientes
Interfaces de usuários de rede

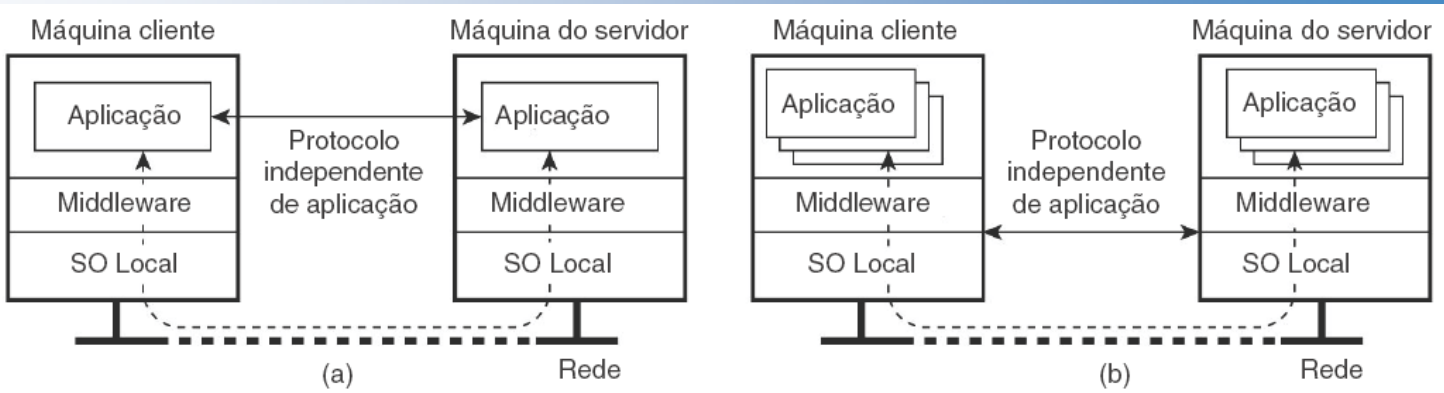


Figura 3.7 (a) Aplicação em rede com seu próprio protocolo. (b) Solução geral para permitir acesso a aplicações remotas.

Clientes
Exemplo: sistema X Window

- Usado para controlar terminais baseados em bits (monitor, teclado, mouse...)

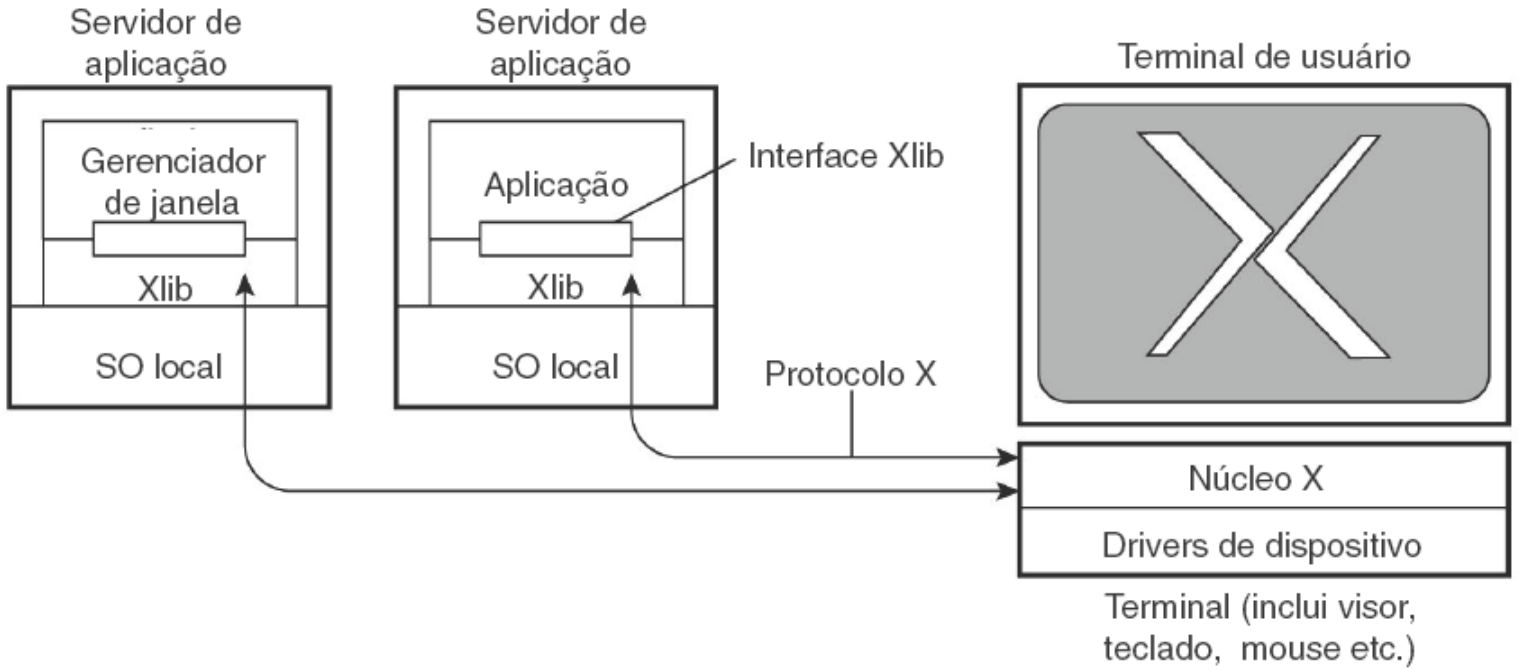


Figura 3.8 Organização básica do sistema X Window.

Clientes
Exemplo: sistema X Window

- Principais problemas
  - Pode ser necessário uso de compactação, por exemplo na reprodução de vídeos ou dados multimídia. Para tanto os terminais devem ser capazes de descompactar os dados, tornando-se mais caros!
  - Em aplicativos fortemente acoplados, haverá grande espera por resposta, pois não haverá tarefas a serem feitas enquanto o servidor processa tarefas.

Clientes
Exemplo: sistema X Window

- Uso de documentos compostos, que integram aplicativos distintos para cooperarem entre si, dividindo a carga entre os servidores que servem os serviços para cada tipo.
  - Arrastar e soltar um documento em outro ou na lixeira;
  - Editar um arquivo de texto que contém uma imagem e um gráfico.

Software do lado cliente para transparência de distribuição

- Parte do nível de processamento e dados são executados no lado do cliente;
  - Caixas automáticos, leitoras de código de barras;
- Transparência de acesso
  - manipulada por meio da geração de um apêndice de cliente conforme uma definição da interface do que o servidor tem a oferecer, ocultando possíveis diferenças em arquitetura de máquina e comunicação propriamente dita.

Software do lado cliente para transparência de distribuição

- Transparência de localização, migração e relocação;
  - Troca de servidores móveis sem conhecimento da aplicação cliente.
- Transparência de replicação;
  - Software do lado cliente envia réplicas de requisição a todos servidores, passando uma única resposta à aplicação cliente.

Software do lado cliente para transparência de distribuição

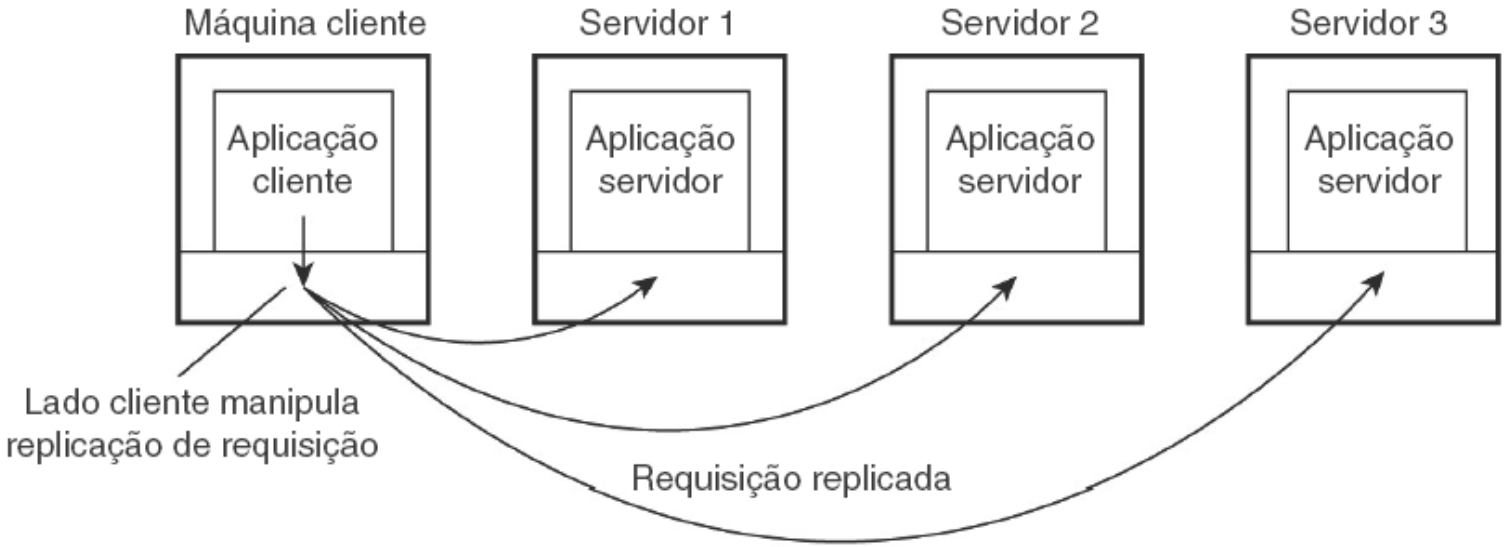


Figura 3.9 Replicação transparente de um servidor usando uma solução do lado cliente.

Software do lado cliente para transparência de distribuição

- Transparência a falha;
  - Mascaramento de falhas: repetir requisições N vezes ao servidor, e tentar trocar em caso de falha.
- Transparência de concorrência;
  - Usa servidores intermediários especiais, e requer menos suporte de software cliente.

Servidores
Questões Gerais de projeto

- Processo que implementa um serviço em nome de um conjunto de clientes. Podem ser organizados de várias formas:
  - Iterativo: manipula a requisição e, se necessário, retorna uma resposta ao cliente requisitante.
  - Concorrente: não manipula por si próprio a requisição, mas passa para um thread separado ou para um outro processo.

Servidores
Questões Gerais de projeto

- Aguardam requisições em:
  - portas específicas, podendo ter suas portas designadas pela IANA (Internet Assigned Numbers Authority);
  - portas designadas dinamicamente pelo sistema operacional, por exemplo, através de um daemon (serviço) ou de um superservidor.

Servidores
Questões Gerais de projeto

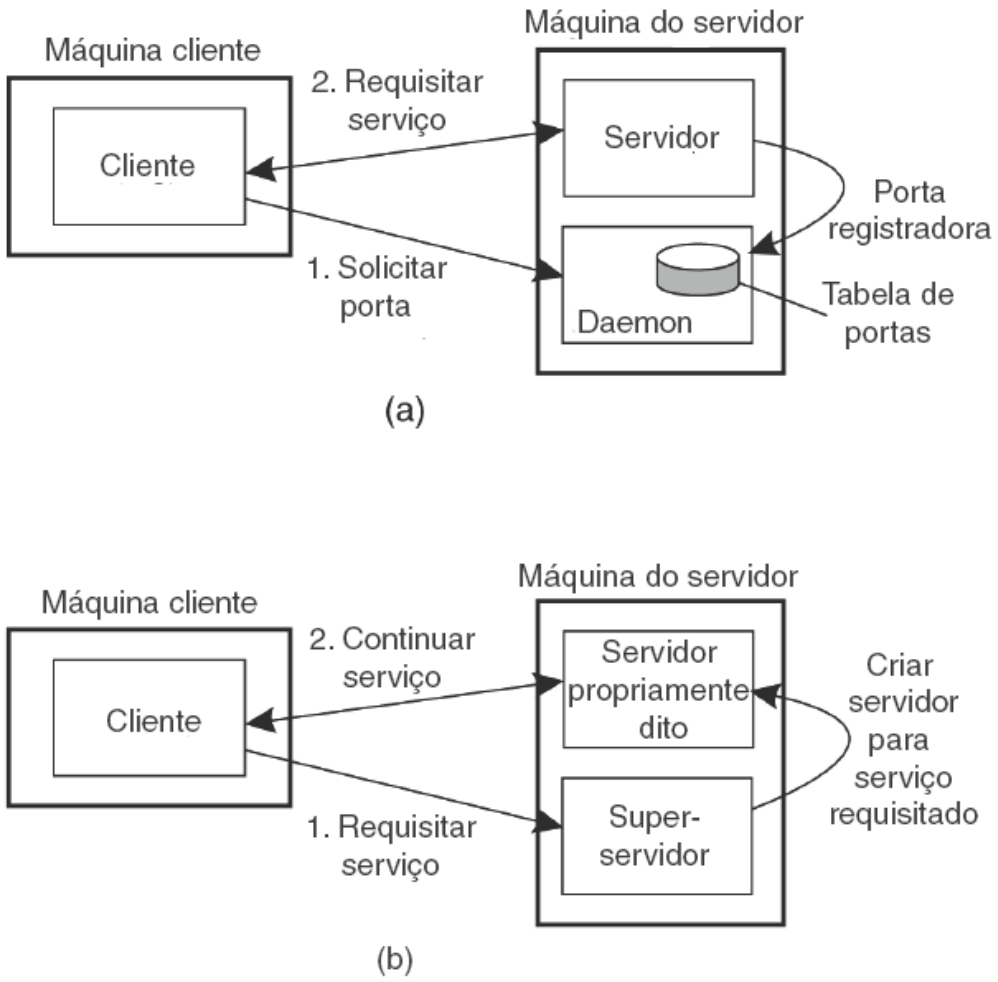


Figura 3.10 (a) Vinculação cliente–a–servidor usando um daemon. (b) Vinculação cliente–a–servidor usando um superservidor.

Servidores
Questões Gerais de projeto

- Interrupção de servidor pode ser feita:
  - Através de finalização forçada da tarefa;
  - Enviar dados de interrupção fora da banda;
- Servidor possui estado?
  - Sem estado: não mantém informações sobre o estado de seus clientes;
  - Estado flexível: mantém dados por tempo limitado;
  - Com estado: mantém informações persistentes sobre o cliente.

Clusters de Servidores
Organização Geral

- Conjunto de máquinas conectadas por uma rede, no qual cada máquina executa um ou mais servidores.
- Normalmente organizado logicamente em três camadas, sendo:
  1. Comutador (lógico) por meio do qual são roteadas as requisições de clientes;
  2. Servidores dedicados a processamento da aplicação;
  3. Servidores de processamento de dados;

Clusters de Servidores
Organização Geral

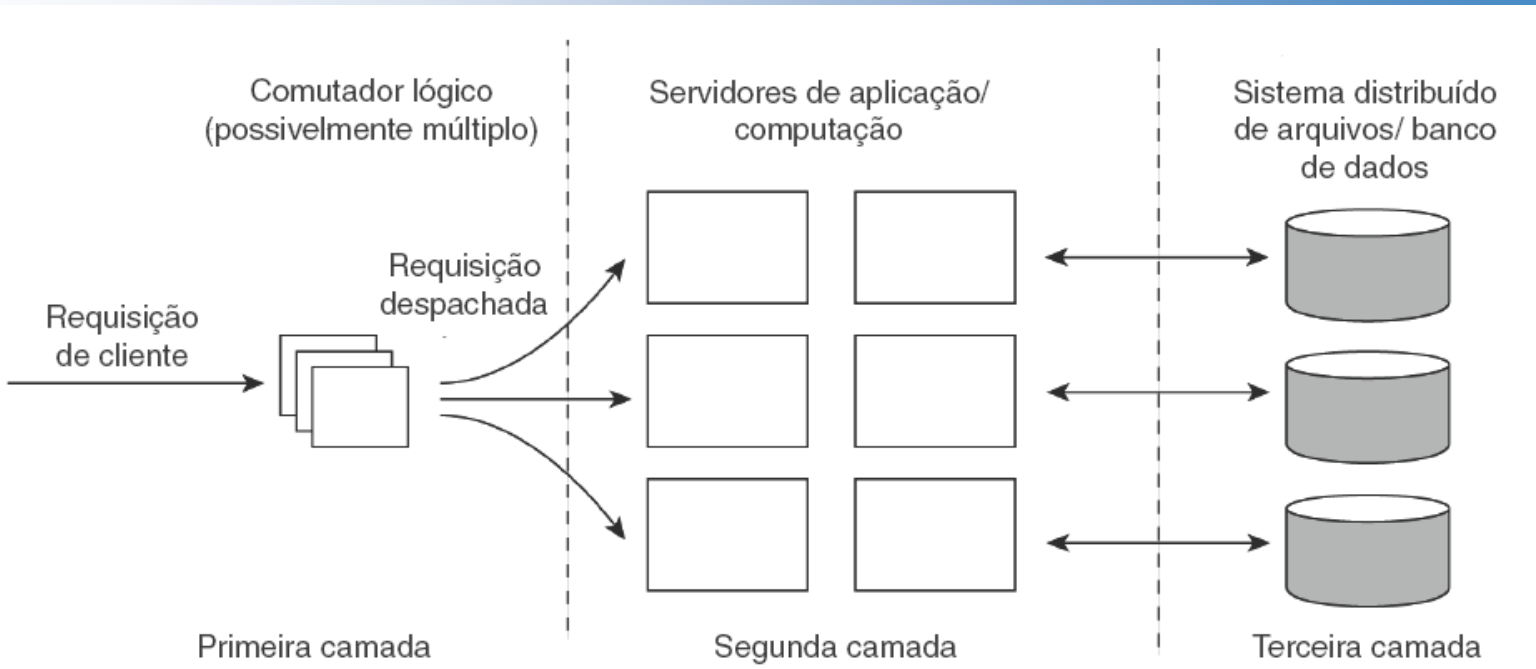


Figura 3.11 Organização geral de um cluster de servidores de três camadas.

Clusters de Servidores
Organização Geral

- Vários serviços distribuídos em vários servidores de aplicação.
  - Comutador deve distinguir serviços para repassar requisições para as máquinas corretas;
  - Carga desbalanceada de servidores (solução: migração de código);

Clusters de Servidores
Transparência de acesso por TCP *Handoff*

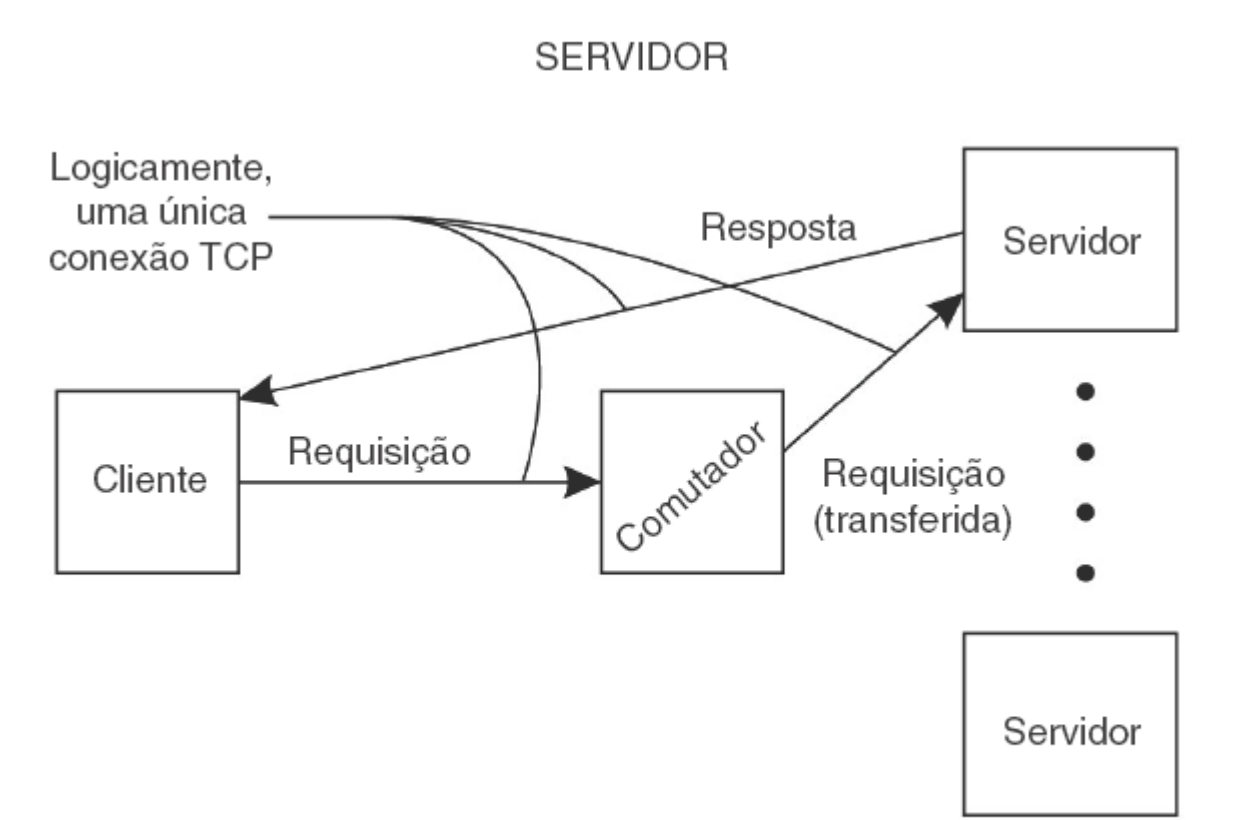


Figura 3.12 Princípio da transferência TCP.

Clusters de Servidores
Servidores distribuídos

- Uso de um único ponto de acesso (por exemplo o comutador) causa a indisponibilidade do cluster inteiro em caso de falha.
  - Solução: fornecer vários pontos (ex.: através de DNS).
  - Ainda requer que clientes façam diversas tentativas em caso de falha, e ainda requer pontos estáticos.

Clusters de Servidores
Servidores distribuídos

- Um servidor distribuído é um conjunto de máquinas que possivelmente muda dinamicamente, com vários pontos de acesso também possivelmente variáveis, que se apresenta ao mundo externo como uma única e poderosa máquina.
  - Uso de MIPv6 auxilia a conseguir um ponto de acesso estável em tal sistema.

Servidores distribuídos
Funcionamento do MIPv6

- Um nó móvel possui:
  - **Rede Nativa**, em que normalmente reside, na qual recebe um endereço estável associado, o **HoA** (Home Address);
  - **Endereço Externo**, ou **CoA** (Care-of Adress), endereço recebido ao se conectar a uma rede externa.
  - Um **agente nativo** faz a transferência dos dados do **HoA** para o **CoA** quando este estiver fora.

Servidores distribuídos
Funcionamento do MIPv6

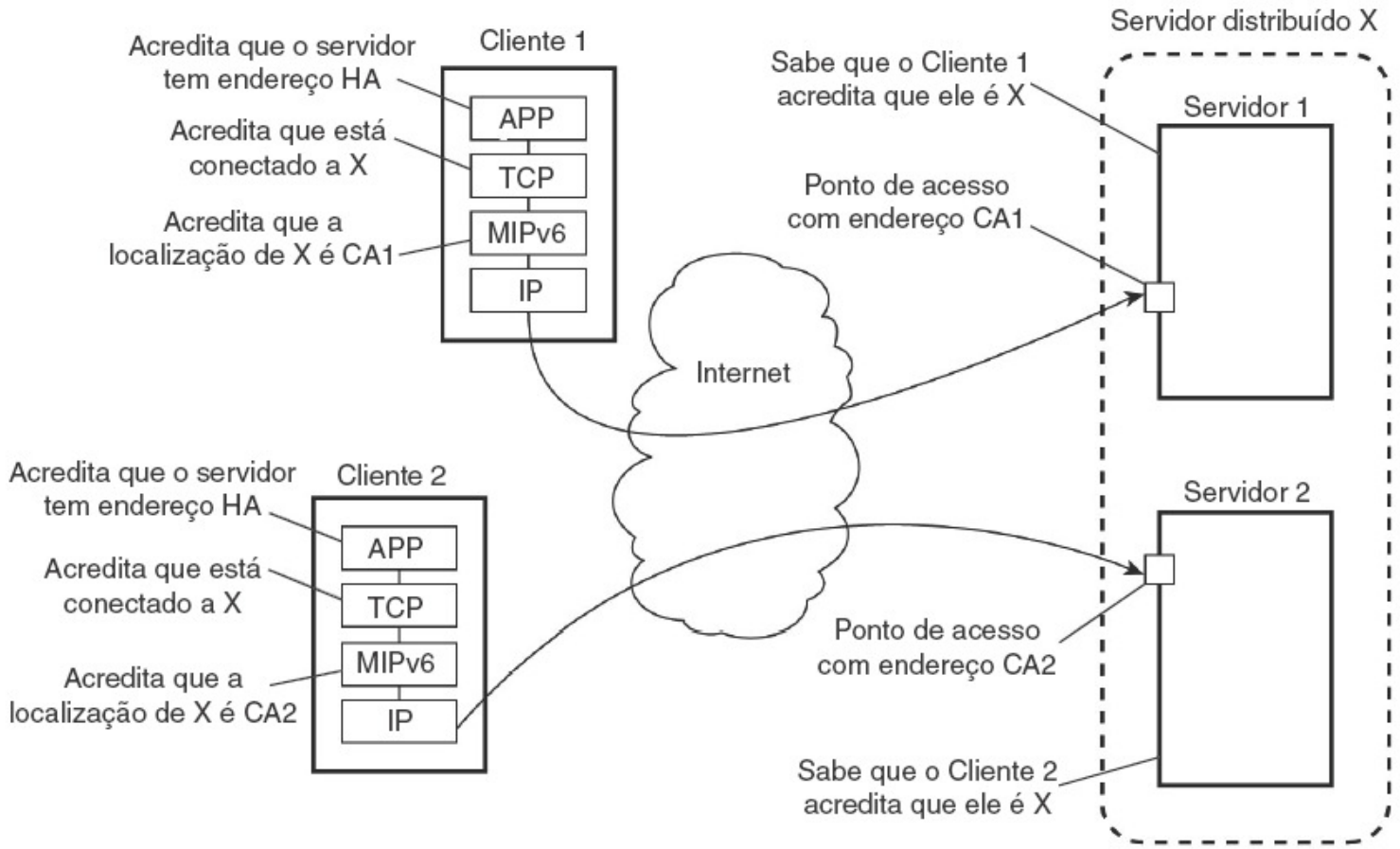


Figura 3.13 Otimização de rota em um servidor distribuído.

Migração de Código

- Torna possível a **migração de processos** entre servidores. Finalidades?
  - Balanceamento de carga;
  - Exploração do paralelismo;
  - Flexibilidade;
  - Configurar dinamicamente sistemas distribuídos.

Migração de Código

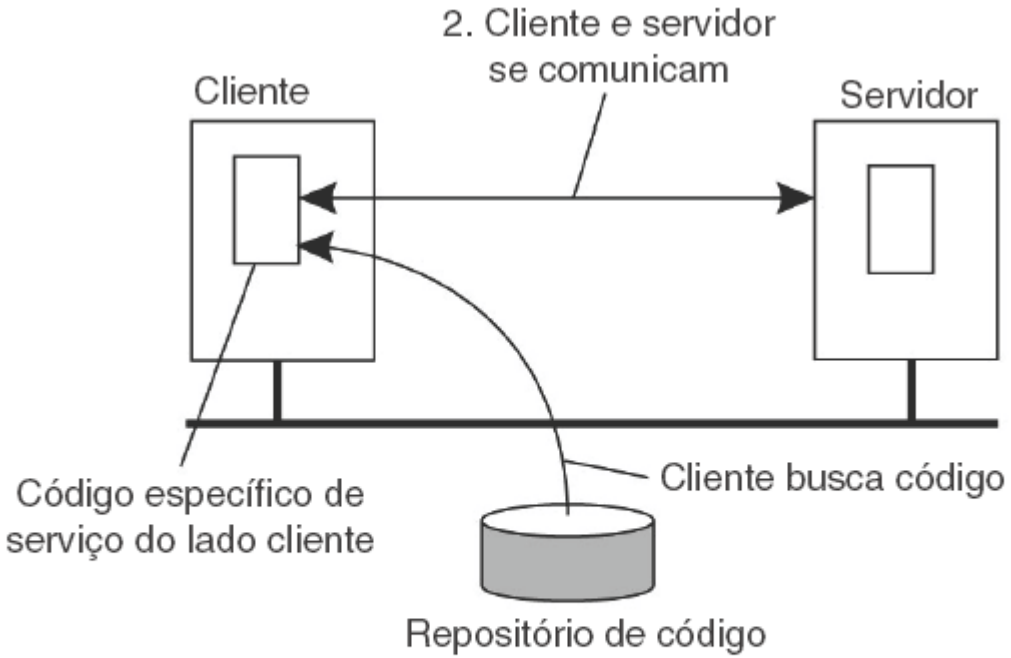


Figura 3.16 Princípio de configuração dinâmica de um cliente para se comunicar com um servidor. Primeiro o cliente busca o software necessário e então invoca o servidor.

Modelos para migração de código

- Estruturas:
  - Segmento de código;
  - Segmento de recursos;
  - Segmento de execução;
- Mobilidade Fraca
  - É possível transferir apenas um segmento de código, talvez com alguns parâmetros para seu início.7
- Mobilidade Forte
  - O segmento de execução também pode ser transferido.

Modelos para migração de código

- A migração pode ser iniciada por:
  - Remetente: iniciado pela máquina em que o código está em execução no momento;
    - Requer autenticação do usuário;
    - Requer proteção de recursos.
  - Destinatário: a iniciativa é tomada pela máquina-alvo (ex.: Java Applets).
    - Mais simples que a primeira;
    - Feita apenas para melhorar o desempenho do cliente;
    - Não requer maiores políticas de segurança.

Modelos para migração de código

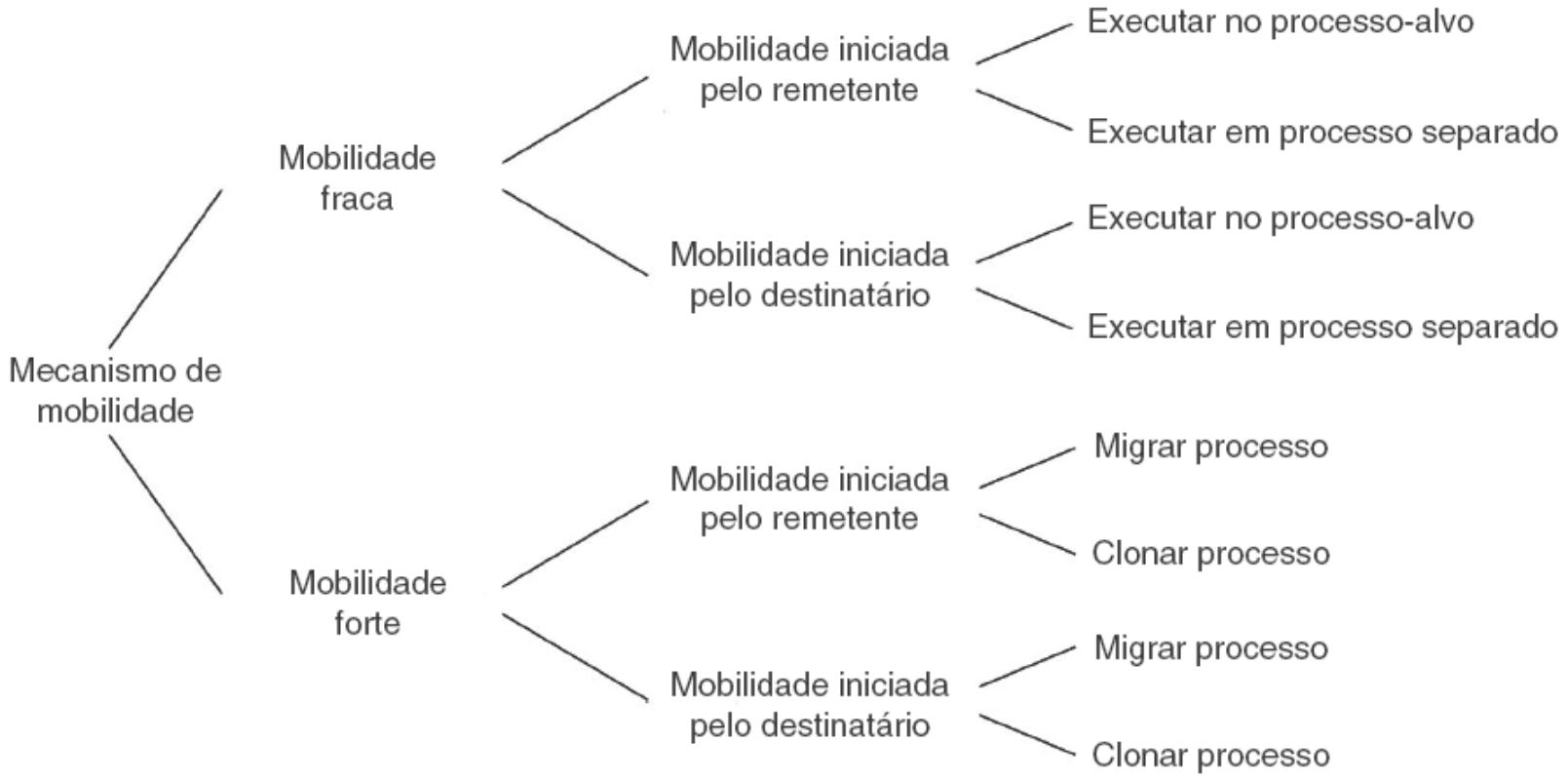


Figura 3.17 Alternativas para migração de código.

Modelos para migração de código
Migração de Recursos Locais

- Mais complicado que migrar código e execução.
- Separadas quanto ao tipo de vinculação e quanto à referência aos recursos.

Modelos para migração de código
Migração de Recursos Locais

- Podem ser vinculados de três formas:
  1. Vinculação por identificador – requer exatamente o recurso referenciado;
  2. Vinculação por valor – só o valor de um recurso é necessário (ex.: bibliotecas padronizadas);
  3. Vinculação por tipo – precisa somente de um recurso de um tipo específico.

Modelos para migração de código
Migração de Recursos Locais

- Podem ser referenciados como:
  1. Recurso não ligado – pode ser movido com facilidade entre máquinas diferentes;
  2. Recurso amarrado – pode ser movido ou copiado, mas com custo relativamente alto;
  3. Recurso fixo – intimamente vinculado a uma máquina ou ambiente específico, não podendo ser movido.

Modelos para migração de código
Migração de Recursos Locais

Vinculação processo–recurso	Vinculação recurso–máquina: Não ligado	Amarrado	Fixo
Por identificador	MV (ou GR)	GR (ou MV)	GR
Por valor	CP (ou MV,GR)	GR (ou CP)	GR
Por tipo	RB (ou MV,CP)	RB (ou GR,CP)	RB (ou GR)

GR	Estabelecer referência global no âmbito do sistema
MV	Mover o recurso
CP	Copiar o valor do recurso
RB	Vincular novamente o processo ao recurso disponível no local

Tabela 3.2 Ações a executar no que se refere às referências a recursos locais quando da migração de código para uma outra máquina.

Migração em sistemas heterogêneos

- Sistemas distribuídos são construídos sobre um conjunto heterogêneo de plataformas, cada um com seu próprio sistema operacional e arquitetura de máquina.
  - O segmento de código é capaz de ser executado num ambiente X-plataforma?
  - Problemas de portabilidade.

Migração em sistemas heterogêneos

- Soluções adotadas:
  - Codificar programas em linguagem de script (PHP, PERL) ou que possam ser executados em máquinas virtuais (ex. Java), para a execução de todo (ou parte do) código;
  - Transferir ambientes inteiros, por exemplo, quando se deseja desligar uma máquina.

Migração em sistemas heterogêneos

Problemas principais:

1. Migrar toda a imagem da memória:
    1. Empurrar páginas de memória para a nova máquina e reenviar as que forem modificadas mais tarde durante a migração do processo;
    2. Parar a máquina virtual corrente, migrar memória e iniciar a nova máquina virtual;
    3. Deixar que a nova máquina virtual puxe novas páginas conforme necessário, isto é, deixar que processos comecem imediatamente na nova máquina virtual e copiar páginas por demanda.

Migração em sistemas heterogêneos

Problemas principais:

2. Migrar vinculações a recursos locais:
    1. Anunciar a nova vinculação rede-endereço MAC, de modo que clientes possam contatar os processos migrados na interface de rede correta;
    2. Se possível admitir que o armazenamento é fornecido como camada separada, migrar vinculação a arquivos também será simples.