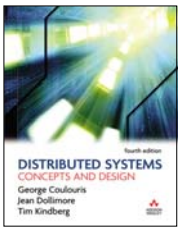
Material baseado no livro *Distributed Systems: Concepts and Design*, 4th Edition, Addison-Wesley, 2005.



2 - Modelos de Sistemas Distribuídos

Agenda:

- Motivação
- Modelos arquitetônicos
- Modelos fundamentais

Motivação

- Sistemas utilizados em ambientes do mundo real devem ser projetados para operar corretamente nas mais variadas circunstâncias e diante de muitas possíveis dificuldades e ameaças
  - Ampla variedade de modos de uso (demanda, requisitos de QoS)
  - Ampla variedade de ambientes e plataformas (desempenho, escala)
  - Problemas internos e ameaças externas (falhas, segurança)
- Propriedades e questões de projeto comuns a diferentes tipos de sistemas distribuídos podem ser melhor estudadas e entendidas na forma de modelos descritivos
  - Cada modelo oferece uma descrição abstrata (ou seja, simples mas consistente com a realidade) de aspectos relevantes do projeto de um sistema distribuído

Para que servem os modelos?

- Capturam a “essência” de um sistema, permitindo que aspectos importante do seu comportamento possam ser mais facilmente estudados e analisados pelos seus projetistas
  - Quais são os principais elementos do sistema?
  - Como eles se relacionam?
  - Que características afetam o seu comportamento (individual *e* colaborativo)?
- Ajudam a:
  - Tornar explícitas todas as pré-suposições sobre o comportamento esperado do sistema
  - Fazer generalizações sobre o que deve e não deve acontecer com o sistema, dadas essas pré-suposições

Tipos de modelos

- Modelos arquitetônicos descrevem a estrutura organizacional dos componentes do sistema
  - Como interagem uns com os outros
  - Como são mapeados para a infra-estrutura física (rede) subjacente
- Modelos fundamentais descrevem problemas e características chaves comuns ao projeto de todos os tipos de sistemas distribuídos
  - Mecanismos de interação e comunicação
  - Tratamento de falhas
  - Segurança

Agenda

- Motivação
- Modelos arquitetônicos
- Modelos fundamentais

Modelos arquitetônicos

- Foco na arquitetura – estrutura de alto nível do sistema, descrita em termos de componentes separadamente especificados e seus relacionamentos
  - Arcabouço de referência para o projeto
  - Base para garantir que a estrutura do sistema atenderá sua atual e provável futura demanda em termos de atributos de qualidade como confiabilidade, adaptabilidade, desempenho, gerência, etc.
- Descrição simplificada e abstrata dos componentes do sistema:
  - Funcionalidades (ou responsabilidades)
    - Ex.: servidor, cliente, *peer*
  - Distribuição física (recursos e carga de trabalho)
    - Ex.: regras de particionamento e/ou replicação
  - Padrões de interação e comunicação
    - Ex.: cliente-servidor, ponto-a-ponto

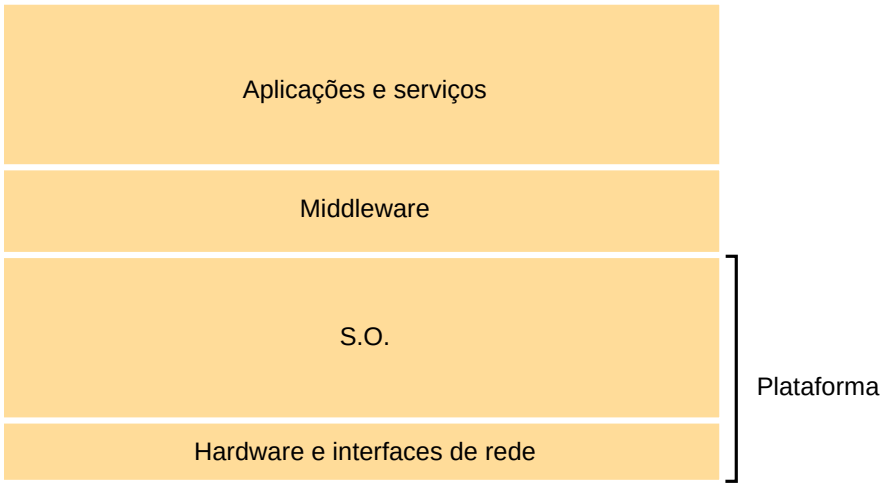
Modelos arquitetônicos

- Classificados e estudados de acordo com as suas características comuns (“estilos arquitetônicos”):
  - Camadas
  - Cliente-servidor
  - Ponto-a-ponto
- Foco na divisão de responsabilidades entre os componentes do sistema e na alocação física desses componentes à infra-estrutura de rede
  - Forte influência nos atributos de qualidade do sistema!
- Na prática, um mesmo sistema distribuído pode apresentar características de diferentes estilos (arquiteturas híbridas)

Estilo Camadas

- Múltiplas camadas de serviços oferecidos e/ou requisitados por processos executando em um ou mais computadores
  - Baseado na estrutura em camadas (hierarquia de módulos) proposta originalmente para sistemas centralizados
- Um serviço pode ser oferecido por um conjunto de servidores que cooperam entre si para manter um visão global consistente do serviço para seus clientes
  - Ex.: Serviço de Tempo da Internet (ITS)
- Duas camadas básicas:
  - Plataforma (Hardware + S. O.)
  - Middleware

Estilo Camadas



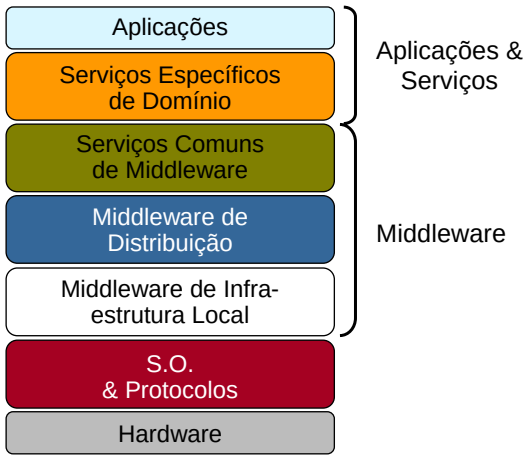
Estilo Camadas

- Plataforma: camada de nível mais baixo que oferece serviços de comunicação e coordenação entre processos para as camadas superiores
  - Implementada de forma independente em cada computador
  - Ex.: Intel/Windows, SunSparc/Solaris, PowerPC/MacOS, Intel/Linux
- Middleware: camada intermediária que oferece abstrações de alto nível para facilitar a comunicação e o compartilhamento de recursos entre os elementos da camada de aplicação
  - Implementada de forma independente por cada fabricante
  - Ex.: RPC, CORBA, DCOM, Java-RMI, EJB, Serviços Web, .NET
  - Uso é opcional (por quê?)

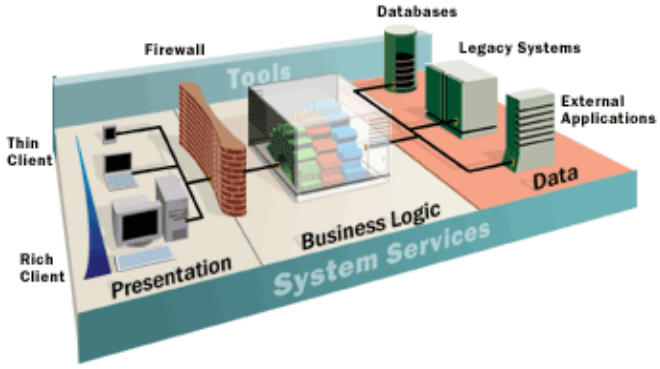
Estilo Camadas

- Limitações da camada de middleware:
  - "Algumas funcionalidades relacionadas à comunicação entre processos remotos só podem ser completamente implementadas, de forma confiável, com o conhecimento e a ajuda das aplicações envolvidas nos dois (ou mais) extremos da comunicação. Portanto, prover essas funcionalidades como um serviços do próprio sistema de comunicação (middleware) nem sempre é a melhor solução." [Saltzer *et al.* 1984]
  - Exemplo: envio de mensagens de correio eletrônico implementado diretamente sobre TCP/IP
    - Desafios?
    - Solução?

Estilo Camadas – Tendência atual



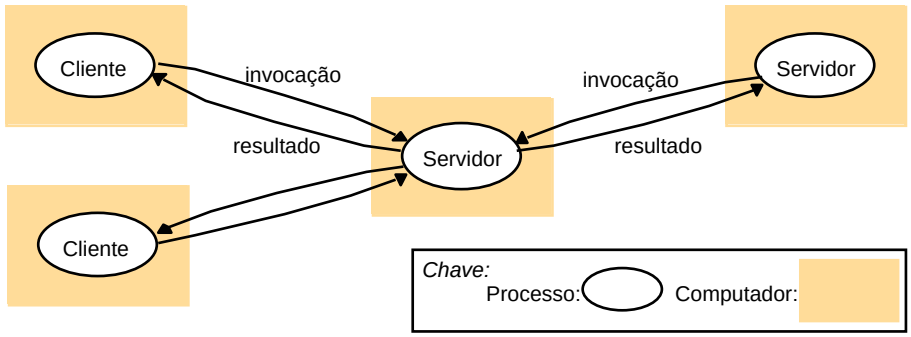
Estilo Camadas no contexto do estilo Cliente-Servidor



Estilo Cliente-Servidor

- Divisão das responsabilidades entre os componentes do sistema de acordo com dois papéis bem definidos:
  - Clientes
  - Servidores
- Servidores são responsáveis por gerenciar e controlar o acesso aos recursos mantidos no sistema
- Clientes interagem com servidores de modo a terem acesso aos recursos que estes gerenciam
- Alguns servidores podem assumir o papel de clientes de outros servidores
  - Ex.: Servidor web no papel de cliente de um servidor de nomes
- Continua sendo o modelo de sistema distribuído mais estudado e utilizado na prática!

Estilo Cliente-Servidor



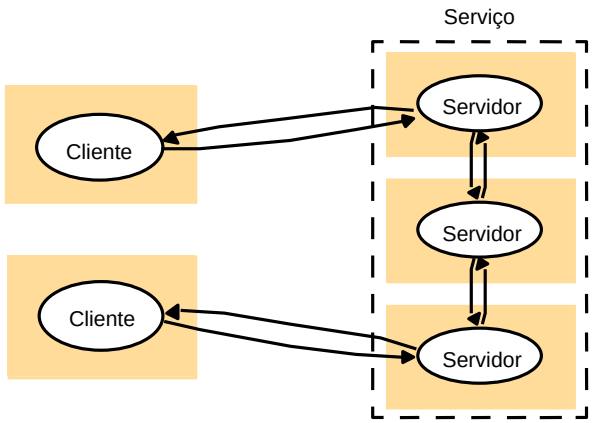
Estilo Cliente-Servidor

- Oito variações do estilo Cliente-Servidor são estudadas neste curso:
  - Múltiplos servidores por serviço
  - Cache e servidores proxy
  - Clientes magros
  - Código móvel
  - Agentes móveis
  - Objetos distribuídos
  - Dispositivos móveis

CS com múltiplos servidores por serviço

- Cada serviço é implementado por um conjunto de servidores, possivelmente localizados em diferentes pontos da rede
- Servidores podem interagir entre si para oferecer uma visão global consistente do serviço para os clientes
- Técnicas mais utilizadas:
  - Particionamento – distribuição física dos recursos entre os vários servidores
    - Maior facilidade de gerência e maior escalabilidade
    - Ex.: Clusters de servidores do portal UOL
  - Replicação – manutenção de cópias do mesmo recurso lógico em dois ou mais servidores
    - Maior desempenho e disponibilidade
    - Ex.: Base de dados do Google, Serviço de nomes da Sun (NIS/NFS)

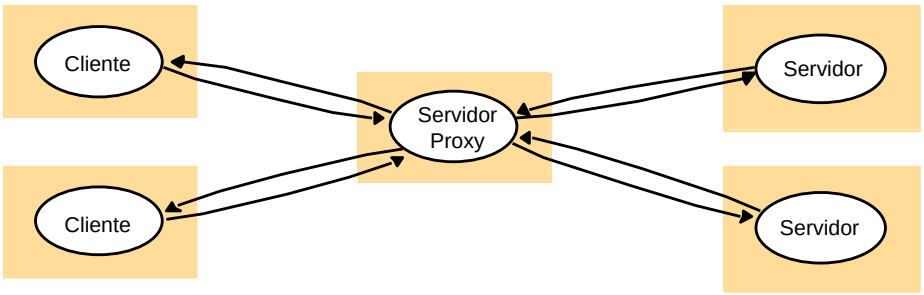
CS com múltiplos servidores por serviço



CS com cache e servidores *proxy*

- Cache
  - Repositório de cópias de objetos recentemente utilizados que está fisicamente mais próximo do que os objetos originais
  - Principais desafios:
    - Política de atualização (controla a entrada e saída de objetos no cache)
    - Localização física (nos clientes ou em um ou mais servidores *proxy*)
- Servidor *proxy*
  - Processo compartilhado por vários clientes que serve como cache para os recursos disponibilizados por outros servidores remotos
  - Principais funções:
    - Reduzir o tempo de acesso
    - Aumentar a disponibilidade
    - Também utilizado para proteção, filtragem, adaptação, etc.

CS com cache e servidores *proxy*

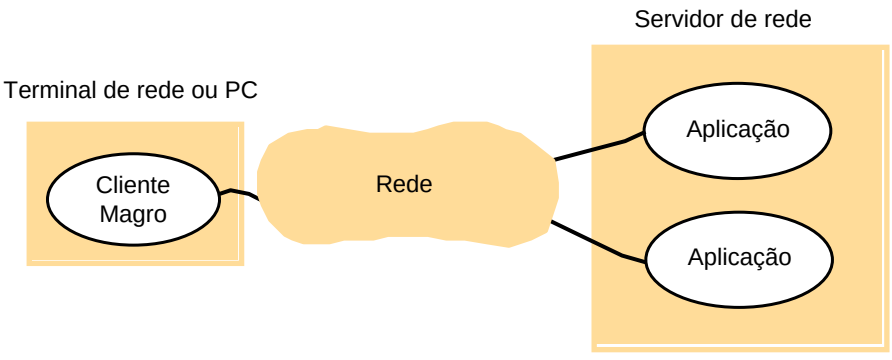


- Pode ser contraproducente! (Em quais circunstâncias?)

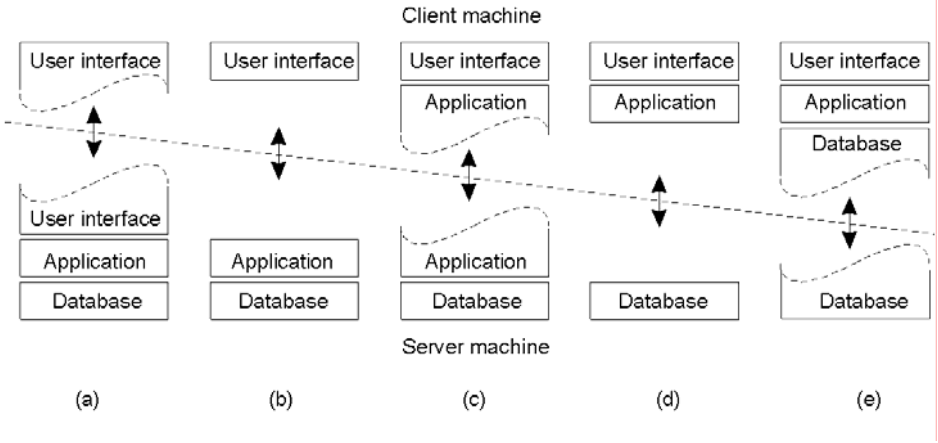
CS com clientes magros

- Cliente magro
  - Camada de software com suporte para interação local com o usuário, e que executa aplicações e solicita serviços exclusivamente a partir de servidores remotos
    - Ex.: XWindows (Unix/Linux), WinFrame (WindowsNT), VNC
  - A favor:
    - Baixo custo de hardware e software para os clientes
    - Maior facilidade de gerência e manutenção das aplicações
  - Contra:
    - Alto custo de hardware e software para os servidores
    - Centralização da carga de trabalho e do tráfego de mensagens
    - Risco de sobrecarga dos servidores e/ou da rede
    - Baixo desempenho para aplicações altamente interativas

CS com clientes magros



Classificação de clientes e servidores quanto ao nível de “gordura” das camadas local e remota



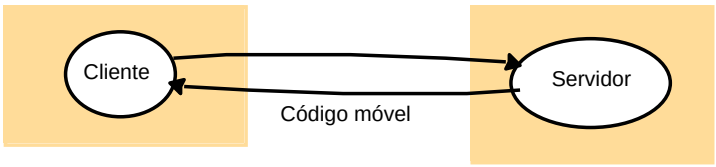
Fonte: Tanenbaum & van Steen 2002

CS com código móvel

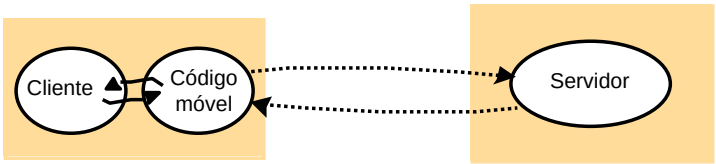
- Serviços oferecidos na forma de um código (programa) específico que deve ser descarregado do servidor
  - Aplicações clientes executam e interagem localmente com o código móvel recebido
  - Dependendo do serviço, código móvel pode interagir com um ou mais servidores em nome da aplicação cliente
  - Ex.: Java applets, Tcl scripts
- Principais benefícios:
  - Redução do tempo de resposta para aplicações interativas
  - Maior facilidade de customização e atualização da interface de acesso ao serviço
  - Possibilidade de estender dinamicamente as funcionalidades das aplicações clientes

CS com código móvel

a) Cliente requisita o serviço baixando o código móvel do servidor



b) Cliente executa e interage localmente com o código móvel recebido



c) Código móvel pode interagir com o servidor em nome do cliente

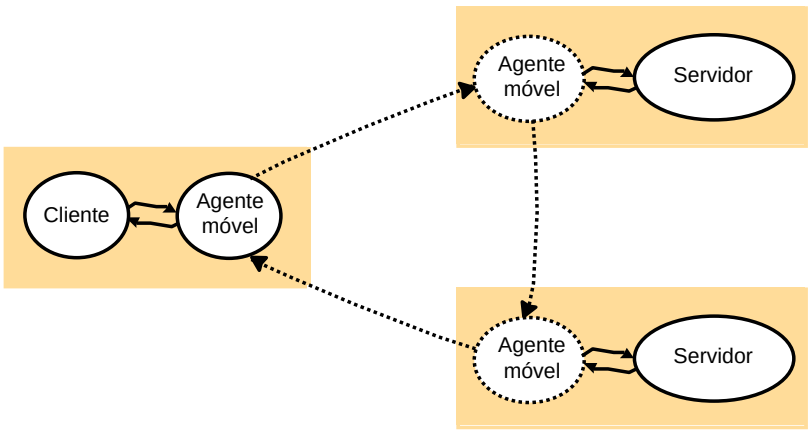
CS com código móvel

- Desafios de projeto:
  - Heterogeneidade do código móvel e da arquitetura de execução das aplicações clientes
    - Solução: máquinas virtuais padronizadas embutidas nas aplicações clientes
  - Riscos de segurança na execução do código móvel
    - Solução: limitar as ações do código móvel ou executá-lo em um ambiente isolado do restante da rede
  - Atrasos causados pelo tempo de transferência do código móvel e pelo tempo de inicialização do seu ambiente de execução
    - Solução: transferência do código em formato compactado; cache de códigos recentemente utilizados; pré-inicialização do ambiente de execução

CS com agentes móveis

- Agente móvel
  - Programas em execução (código + dados) que circula pela rede solicitando serviços em nome de um usuário ou de uma aplicação cliente
    - Ex.: agente para coleta de dados, busca e comparação de preços de produtos, instalação de software, etc
  - O acesso aos serviços é feito localmente pelo agente, ou de locais fisicamente próximos (da mesma rede local) aos servidores
- Benefícios:
  - Redução dos custos e do tempo de acesso
    - Acesso antes remoto agora passa a ser local
  - Maior tolerância a falhas de comunicação
    - Conexão necessária apenas durante a transferência do agente
  - Melhor distribuição do tráfego de mensagens na rede

CS com agentes móveis



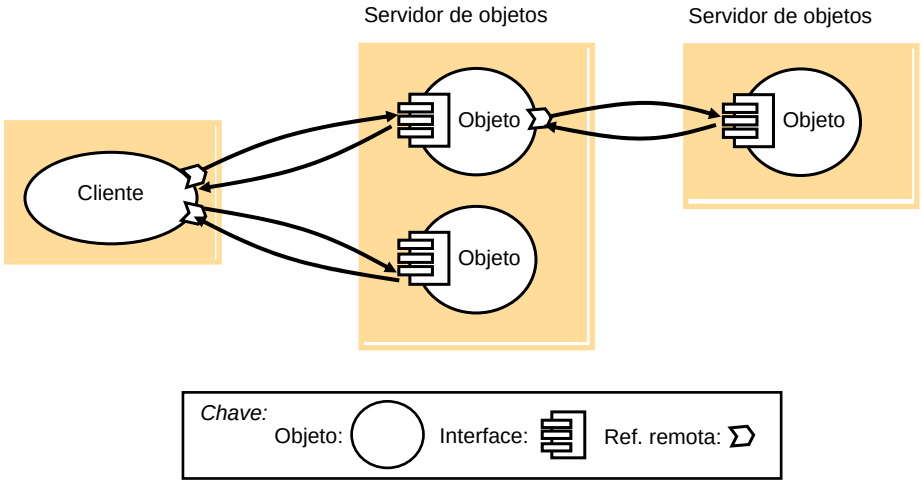
CS com agentes móveis

- Desafios de projeto:
  - Heterogeneidade do código e da arquitetura de execução dos agentes
    - Solução: ambientes de execução padronizados em cada ponto do sistema
  - Riscos de segurança na execução dos agentes
    - Solução: restringir a entrada a agentes certificados e executá-los em um ambiente isolado ou com acesso aos recursos locais rigorosamente controlado
  - Riscos de interrupção dos agentes devido à negação de acesso por parte dos servidores
    - Solução: utilizar agentes certificados e com permissão de acesso aos recursos requisitados
  - Atrasos causados pela tempo de transferência dos agentes
    - Solução: transferência dos agentes em formato compactado

CS com objetos distribuídos

- Objetos encapsulados em processos servidores
  - Objetos acessados por outros processos (clientes) através de referências remotas para uma ou mais de suas interfaces
  - Referência remota permite invocar remotamente os métodos disponíveis na interface do objeto referenciado
- Implementação na forma de middleware orientada a objetos (ex.: CORBA, COM+, Java-RMI, EJB, .NET)
  - Diferentes mecanismos para criar, executar, publicar, localizar, e invocar objetos remotos
  - Diferentes serviços de suporte
    - Transação, persistência, replicação, segurança, etc
  - Diferentes fabricantes e modelos de negócio
- Discutido em detalhes no Cap.5!

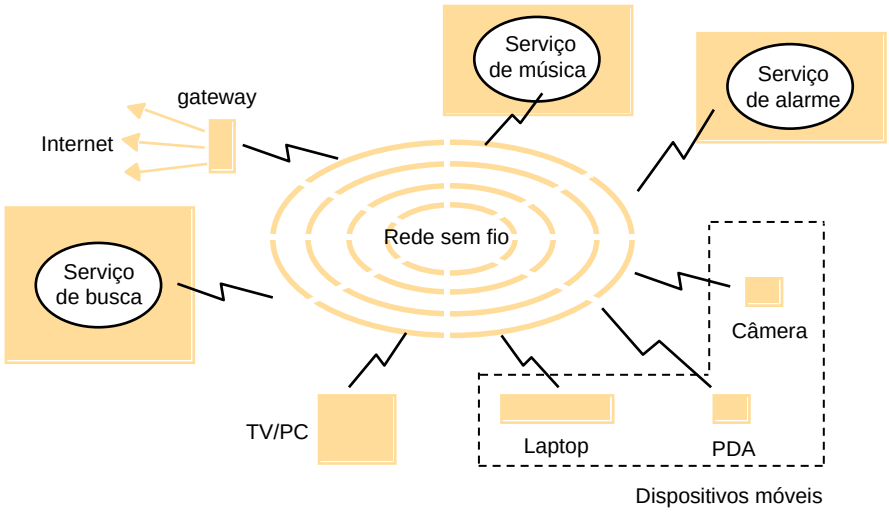
CS com objetos distribuídos



CS com dispositivos móveis

- Formado por aplicações clientes que executam em dispositivos móveis (PDAs, laptops, celulares, etc) e acessam servidores da rede fixa através de uma infra-estrutura de comunicação sem fio
  - Diferença para as variações com código móvel e agentes móveis?
- Principais benefícios:
  - Fácil conexão dos dispositivos a uma nova rede local
    - Inclusão de novos clientes sem a necessidade de configuração explícita
  - Fácil integração dos clientes aos serviços locais
    - Descoberta automática de novos serviços (sem intervenção do usuário)
- Desafios de projeto:
  - Identificação de recursos independente de sua localização física
  - Limitações de processamento, tempo de conexão e largura de banda
  - Privacidade e segurança

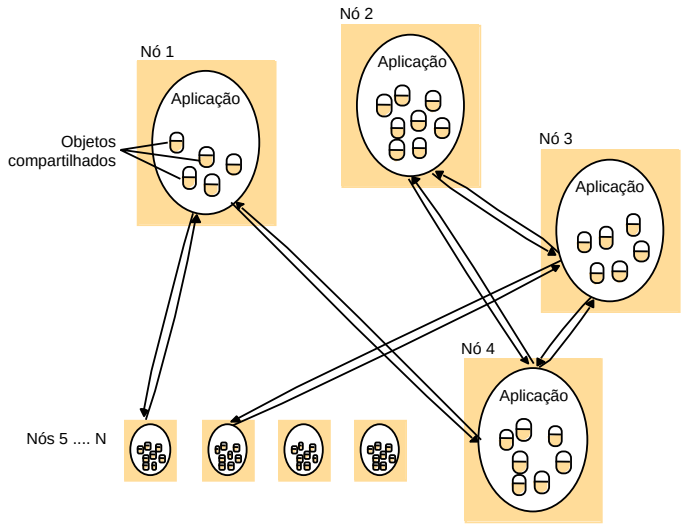
CS com dispositivos móveis



Estilo Ponto-a-Ponto

- Todos os processos (nós) envolvidos em uma mesma tarefa ou atividade exercem papéis similares, interagindo cooperativamente como “parceiros” (*peers*) uns dos outros
  - Criado para suprir as conhecidas deficiências de escalabilidade do modelo CS tradicional
  - Padrão de interação e de compartilhamento de recurso entre os participantes depende inteiramente dos requisitos da aplicação
- O objetivo principal é explorar os recursos (hardware & software) de um grande número de máquinas/usuários interessados em realizar uma determinada tarefa ou atividade
  - Uso pioneiro no compartilhamento de arquivos de áudio (Napster)
  - Sucesso do Napster abriu caminho para vários outros sistemas e middleware P2P de propósito geral (KaZaA, Gnutella, Emule, JXTA, Pastry, etc)

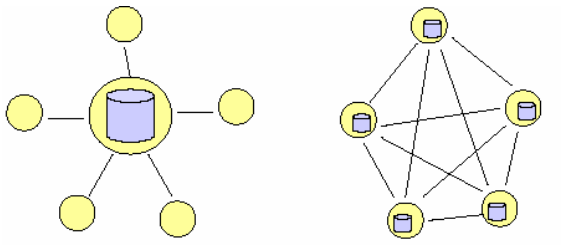
Estilo Ponto-a-Ponto



Estilo Ponto-a-Ponto

- Características dos sistemas P2P:
  - Arquitetura totalmente distribuída (sem qualquer controle centralizado)
  - Sem distinção entre clientes e servidores (cada nó é cliente e servidor ao mesmo tempo)
  - Nós podem trocar recursos diretamente entre si
  - Nós são autônomos para se juntarem ao sistema ou deixá-lo quando quiserem
  - Necessidade de código de coordenação em cada nó para manter a consistência dos recursos e sincronizar as ações em nível de aplicação
  - Interação entre nós pode ser feita utilizando mecanismos apropriados para comunicação em grupo (difusão seletiva, notificação de eventos)
- Discutidos em detalhes no Cap.10 (não abordado no curso!)

Estilo P2P X Estilo Cliente-Servidor



Cliente-Servidor P2P

Benefícios e desafios do estilo P2P

Característica	Benefícios	Desafios	Áreas de aplicação
Troca direta	Eficiência	Segurança	Compartilhamento de dados (Napster), Bate-papo
Cliente & servidor	Compartilhamento de recursos computacionais	Balanceamento de carga	Processamento distribuído (SETI@Home), Colaboração (jogos)
Cada nó como provedor	Recursos massivos	Busca, Integração de dados	Compartilhamento de dados (Napster)
Autonomia	Tolerância a falhas	Disponibilidade de dados, Replicação	Todas

Arquiteturas P2P

- Arquitetura básica (ou “pura”)
  - Nenhum nó é especial
  - Cada nó conhece apenas os seus vizinhos
    - Determinados quando o nó se junta ao sistema (busca na rede pelos vizinhos mais próximos)
    - Conjunto de vizinhos pode mudar ao longo do tempo
  - Topologia estruturada (ex.: Chord) ou não-estruturada (ex.: Gnutella)
- Arquitetura híbrida
  - Algumas tarefas (com descoberta da localização de recursos) realizadas através de um ou mais componentes centralizados
  - Demais tarefas realizadas de forma descentralizada através da interação direta entre os nós interessados
  - Exs.: Napster, KaZaA

Descoberta de recursos nas arquiteturas P2P

- Mecanismos de busca na arquitetura básica
  - Topologia não estruturada
    - Busca por inundação
    - Busca cega
    - Busca informada
    - Busca informada com replicação
  - Topologia estruturada
    - Busca via mapeamento de IDs
- Mecanismos de busca na arquitetura híbrida
  - Busca em catálogo centralizado
  - Busca via super nós

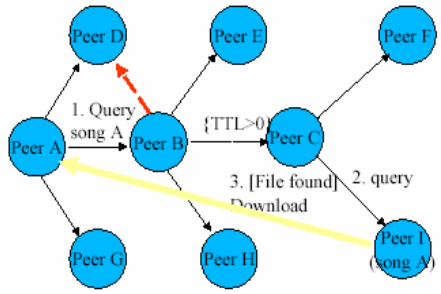
Busca por inundação

- O nó que inicia uma consulta a envia para todos os seus vizinhos
- Ao receber uma consulta:
  - Se o nó possui o recurso, ele notifica o iniciador da consulta; o iniciador da consulta pode então obter o recurso diretamente do nó que o notificou
  - Se o nó não possui o recurso, ele decrementa o valor do tempo de vida (*Time To Live – TTL*) da consulta e, caso TTL > 0, repassa a consulta para todos os outros vizinhos

Busca por inundação (cont.)

- Observações:
  - O iniciador da consulta pode obter resultados redundantes ou até não obter nenhum resultado mesmo que o recurso exista em algum nó da rede (Por quê?)
  - Um nó pode ser visitado mais de uma vez (nó “D” no exemplo a seguir)
  - ID do iniciador pode ser transmitido junto com a consulta ou ID do provedor do recurso pode ser informado ao iniciador seguindo o caminho inverso da consulta

Exemplo da busca por inundação



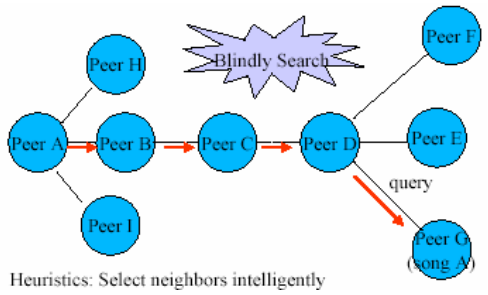
Desvantagens da busca por inundação

- Número grande de mensagens trocadas entre os nós
- Consultas enviadas de forma duplicada
- Difícil escolher o valor do tempo de vida das consultas
  - TTL alto demais pode sobrecarregar a rede
  - TTL baixo demais pode encerrar a busca antes de chegar ao provedor do recurso

Busca cega (ou busca aleatória)

- Iniciador da consulta seleciona um único vizinho para enviar a consulta
- A seleção é baseada em algumas heurísticas:
  - Por exemplo, se um vizinho sempre retorna um resultado satisfatório, ele deve ser selecionado com mais freqüência
    - Deve ser possível saber quando cada vizinho foi capaz ou não de encontrar um determinado recurso
- Se o vizinho não possui o recurso, ele seleciona um dos seus vizinhos para repassar a consulta
  - Seleção também baseada em heurísticas
- O processo se repete até que o recurso seja encontrado ou o tempo de vida consulta expire (TTL = 0)

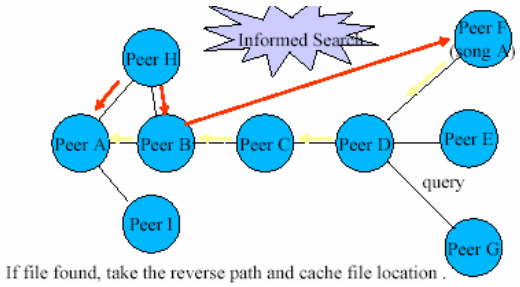
Exemplo da busca cega



Busca informada

- Cada nó possui um *cache* para armazenar a localização de recursos que tenham sido consultados previamente
- Ao receber uma consulta:
  - Se o nó encontra a localização do recurso no seu *cache*, ele a informa ao nó de quem recebeu a consulta
  - Do contrário, ele tenta descobrir a localização do recurso fazendo busca por inundação
- Uma vez que o recurso é encontrado, o caminho inverso da consulta é usado para informar a localização ao iniciador da consulta
  - Dessa forma, os nós que fazem parte do caminho percorrido pela consulta podem atualizar seus *caches*, acelerando assim as próximas consultas

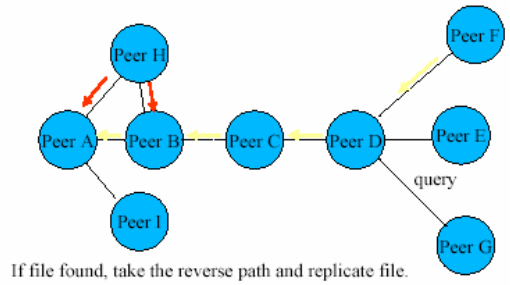
Exemplo da busca informada



Busca informada com replicação

- A diferença para a busca informada simples (sem replicação) é que os *caches* dos nós que compõem o caminho inverso da consulta serão usados para armazenar o próprio recurso, ao invés de apenas a sua localização
  - Dessa forma, os nós que fazem parte do caminho da consulta acelerarão não apenas a descoberta mas também o próprio acesso ao recurso nas próximas consultas

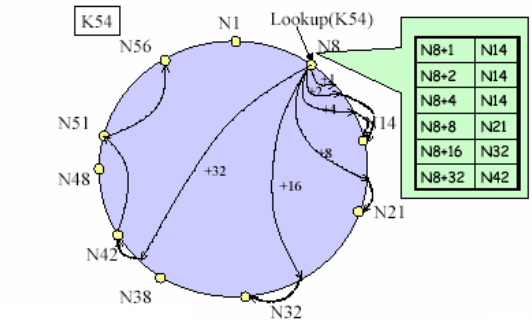
Exemplo da busca informada com replicação



Busca via mapeamento de IDs

- Os recursos são distribuídos através dos nós de acordo com um algoritmo de mapeamento
  - Os nós não escolhem seus recursos por “vontade própria”
  - Necessidade de um mecanismo de replicação adicional para oferecer maior disponibilidade
- Mecanismo de mapeamento
  - Cada nó tem um identificador único (ex.: Hash ou IP)
  - Cada recurso tem uma chave de busca
  - Cada nó é responsável por armazenar recursos que tenham chaves de busca “similares” ao (ou seja, que possam ser facilmente mapeadas para o) identificador do nó
  - Dada uma consulta com uma chave de busca, qualquer nó é capaz de repassá-la rapidamente ao nó cujo identificador mais se assemelha à chave

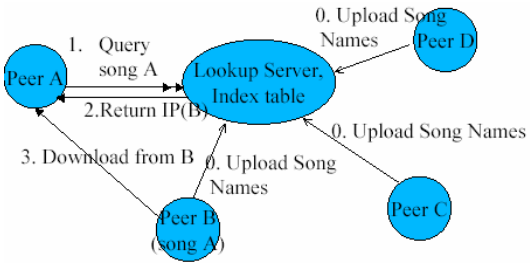
Exemplo da busca via mapeamento de IDs



Busca em catálogo centralizado

- Descoberta da localização dos recursos feita através de consulta a um único nó central (servidor de *lookup*)
- Cada nó fornece ao servidor de *lookup* meta-informação descrevendo os recursos que provê
- Acesso aos dados dos recursos feito diretamente entre os nós clientes e o nós provedores

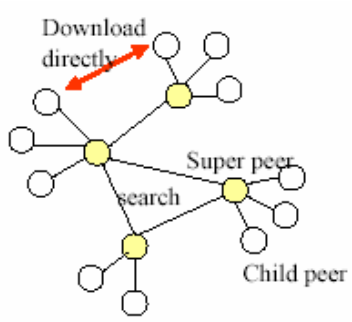
Exemplo da busca em catálogo centralizado



Busca via super nós

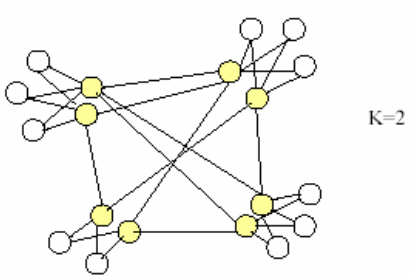
- Nós organizados numa estrutura hierárquica composta por nós “filhos” e “super” nós
- Cada super nó mantém um catálogo descrevendo os recursos mantidos por cada um de seus nós filhos
- Um nó filho inicia uma consulta a enviando para o seu super nó
- Um super nó processa consultas recebidas de outros super nós em nome de seus nós filhos
- Acesso aos dados dos recursos feito diretamente entre os nós clientes e os nós provedores

Exemplo da busca via super nós



Desafios da busca via super nós

- Proporção entre super nós e nós folhas?
- Topologia entre super nós?
  - Estruturada ou não estruturada?
- Busca mais eficiente?
- Sistema mais confiável?
  - K-redundância?



Comparação entre os mecanismos de descoberta

Característica	**Topologia / Mecanismo de busca**			
	Híbrida	**Não estruturada**		**Estruturada**
		Cega	**Informada**	
Estrutura	Centralizada	Aleatória	Aleatória	Fixa
Método de busca	Catálogo centralizado	Inundação	Inundação (opcional)	Mapeamento de IDs
Informação sobre localização	Determinística	Nenhuma	Parcial	Alta probabilidade
Distribuição dos recursos	Qualquer lugar	Qualquer lugar	Qualquer lugar	Fixa
Re-configuração dos nós	Gargalo	Boa	Boa	Ruim

Requisitos de projeto

- Questões chave para o projeto (arquitetura) de um sistema distribuído
  - Desempenho
  - Qualidade de serviço (QoS)
  - Cache e replicação
  - Confiabilidade
- Geralmente consideradas na implementação de aplicações distribuídas que compartilham recursos em larga escala

Desempenho

- Capacidade do sistema para reagir de forma rápida e consistente às requisições dos usuários
  - Sujeita às limitações de processamento e de comunicação dos computadores e da infra-estrutura de rede
- Principais fatores envolvidos:
  - Tempo de resposta (*responsiveness*)
    - Afetado pelo número de camadas de software necessário para a invocação dos serviços remotos e pelo volume de dados transferidos através da rede
  - Taxa de trabalho (*throughput*)
    - Medida do desempenho do sistema considerando todos os usuários
  - Balanceamento de carga (*load balance*)
    - Utilizado para explorar de forma mais eficiente os recursos computacionais disponíveis

QoS

- Capacidade do sistema para oferecer serviços com garantias suficientes para atender de forma satisfatória as necessidades específicas de seus usuários
- Principais fatores:
  - Confiabilidade
  - Segurança
  - Desempenho
  - Adaptabilidade
  - Disponibilidade
- Aspectos de confiabilidade, segurança e desempenho serão abordados no contexto dos modelos fundamentais de falha, segurança e interação, respectivamente

QoS

- No contexto de QoS, o desempenho também é definido em termos da capacidade do sistema de atender restrições de tempo crítico
  - Ex.: sistemas de tempo real, sistemas para transmissão de dados multimídia
- Em geral, essas restrições devem ser mantidas durante todo o tempo, e sob todas as circunstâncias, em que os recursos são utilizados, especialmente sob alta demanda
  - Recursos críticos devem ser reservados a priori junto aos seus respectivos servidores (solicitações não atendidas são rejeitadas)
  - Garantias negociadas entre as partes através de acordos em nível de serviço (SLAs)
- Cap.17 discute em detalhes diversos mecanismos de QoS

Cache e replicação

- Capacidade do sistema para manter múltiplas cópias de um mesmo recurso lógico fisicamente distribuídas, de modo a reduzir o seu tempo de acesso
  - Ex.: protocolo de cache da web
- Principais questões envolvidas:
  - Alocação e distribuição das réplicas
  - Políticas de acesso e atualização
  - Mecanismo de validação
  - Compromisso entre a consistência e a qualidade do serviço
    - Freqüência de atualização X Desempenho
    - Suporte para operações “desconectadas”

Confiabilidade

- Capacidade do sistema para continuar operando efetivamente, mesmo diante da ocorrência de falhas e da ameaça de acessos indevidos aos recursos compartilhados
- Principais questões:
  - Tolerância a falhas
    - Obtida através da redundância (replicação) de recursos lógicos e físicos
    - Implica em maiores custo e complexidade
  - Segurança
    - Obtida através de mecanismos de criptografia, garantia da integridade dos dados, assinatura digitais, políticas de controle de acesso, etc

Agenda

- Motivação
- Modelos arquitetônicos
- Modelos fundamentais

Modelos fundamentais

- Foco em três importantes aspectos de projeto:
  - Mecanismo de interação
  - Tratamento de falhas
  - Segurança
- Utilizados para ajudar a planejar, entender e analisar o comportamento esperado do sistema
- Principais benefícios:
  - Correção antecipada de erros
  - Investigação, avaliação e reuso de diferentes alternativas de projeto
  - Menor custo de desenvolvimento, manutenção e evolução

Modelo de interação

- Descreve as formas de interação e coordenação entre os componentes do sistema
- Influenciado pela:
  - Capacidade de comunicação da rede
    - Latência (transmissão, acesso à rede, S.O.)
    - Largura de banda
    - Instabilidade (*jitter*)
  - Ausência de estado global
    - Impossibilidade de acordo sobre a mesma noção de tempo
    - Dificuldade para sincronizar relógios através da rede
- Duas variantes em relação ao tempo:
  - Modelo síncrono
  - Modelo assíncrono

Modelo síncrono

- Características:
  - O tempo para executar cada passo de um processo tem limites inferior e superior conhecidos
  - Cada mensagem transmitida por um canal de comunicação é recebida dentro de um limite conhecido de tempo
  - Cada processo tem um relógio local cuja taxa de desvio do tempo real tem um limite conhecido
- Vantagens:
  - Mais fácil para programar e analisar o comportamento dos processos
- Desvantagens:
  - Dificuldade de definir e garantir valores realistas para os limites de tempo (*timeouts*)
  - Resultados de análise e simulação podem não ser confiáveis

Modelo assíncrono

- Características:
  - Sem limites conhecidos para
    - Velocidade de execução dos processo
    - Atraso na transmissão das mensagens
    - Taxa de desvio dos relógios
- Vantagens:
  - Mais realista
  - Soluções assíncronas também são válidas para o modelo síncrono
- Desvantagens:
  - Mais difícil de implementar e analisar
- Exemplos?

Problema dos generais bizantinos

- Metáfora comumente utilizada para ilustrar problemas de coordenação e sincronização em sistemas distribuídos
  - Proposto originalmente por Lamport (1982)
  - O livro oferece uma versão bastante simplificada do problema original, com o nome de “Agreement in Pepperland”
- Contexto do problema:
  - Duas divisões do exército bizantino estão acampadas no topo de dois morros próximos à cidade de Bizâncio
  - Mais longe, no vale entre os dois morros, estão posicionados a horda de bárbaros que planejam tomar a cidade
  - As duas divisões estão seguras enquanto permanecerem nos seus acampamentos; além disso, a única chance das duas divisões derrotarem os bárbaros é atacando de forma conjunta

Problema dos generais bizantinos

- Contexto do problema (cont.):
  - As duas divisões podem se comunicar de forma confiável, através do envio de mensageiros uma à outra
  - Diferentes restrições se aplicam sobre a entrega das mensagens, dependendo do modelo de interação considerado
    - Modelo assíncrono: o tempo de entrega das mensagens é indeterminado (por exemplo, pode demorar de alguns minutos a vários dias)
    - Modelo síncrono: os limites mínimo e máximo para o tempo de entrega das mensagens são conhecidos pelas duas divisões
- Questões:
  - P1: Como decidir qual divisão deve liderar o ataque?
  - P2: Como decidir quando cada divisão deve atacar?

Problema dos generais bizantinos

- Considerações sobre as soluções:
  - P1 pode ser resolvida mesmo no modelo assíncrono (por exemplo, cada divisão envia à outra o número de seus membros; lidera quem tiver o maior exército; em caso de empate, uma divisão teria precedência prévia sobre a outra)
  - P2 não pode ser resolvido de forma satisfatória no modelo assíncrono
    - Por exemplo, o que aconteceria se uma das divisões enviasse à outra a seguinte mensagem: “Atacar Imediatamente!”?
  - P2 pode ser resolvido de forma aproximada no modelo síncrono, tirando proveito das restrições de tempo estabelecidas para a entrega das mensagens
    - Exemplo?

Ordenação de eventos

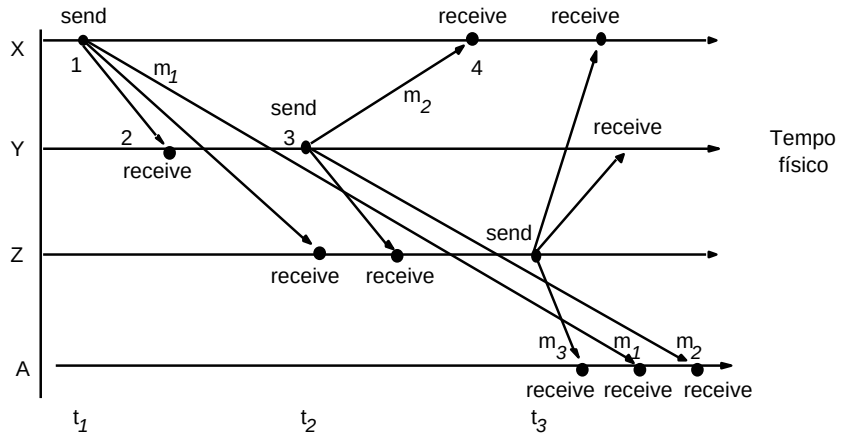
- Em muitos casos, a execução de um sistema distribuído pode ser descrita em termos dos eventos ocorridos no sistema e da ordem em que eles ocorreram
  - Problema: como saber se um determinado evento (ex.: envio ou recebimento de uma mensagem) de um determinado processo ocorreu antes, após ou concorrentemente a um outro evento de um outro processo (possivelmente remoto)?
- Como exemplo, considere o seguinte cenário de troca de mensagens entre um grupo de quatro usuários (X, Y, Z e A) de correio eletrônico
  1. O usuário X envia uma mensagem *m1* com o assunto “Encontro”;
  2. Os usuários Y e Z respondem enviando as mensagens *m2* e *m3*, respectivamente, com o assunto “Re: Encontro”.

Ordenação de eventos

- (Continuação do exemplo)
  - Em tempo “real”, X envia *m1* primeiro; em seguida, Y recebe e lê *m1*, e então responde enviando *m2*; por fim, Z recebe e lê tanto *m1* quanto *m2*, e responde enviando *m3*
  - Porém, devido a atrasos na rede, as três mensagens podem ser entregues a alguns usuários na ordem errada. Por exemplo, o usuário A poderia receber as mensagens na seguinte ordem:

Caixa de Entrada de A	
De	**Assunto**
Z	Re:Encontro
X	Encontro
Y	Re:Encontro

Ordenação de eventos



Ordenação de eventos

- Tentativa de solução: embutir a hora local do processo remetente em cada mensagem enviada. Assim, as mensagens recebidas poderiam ser ordenadas pelo processo de destino, com base no valor da hora embutido em cada uma
  - Problemas?
- Como é impossível sincronizar relógios perfeitamente em um sistema distribuído, Lamport (sempre ele!) propôs a noção de tempo lógico como mecanismo de ordenação de eventos em sistemas distribuídos
  - O conceito de tempo lógico permite estabelecer uma ordem parcial entre eventos ocorridos em processos executando em diferentes computadores, sem a necessidade de recorrer a relógios “reais”
- Discutido em detalhes no Cap.11 (não abordado no curso)!

Modelo de falha

- Descreve as maneiras através das quais podem ocorrer falhas nos processos e canais de comunicação, de modo a facilitar o entendimento dos seus efeitos no sistema
- Tipos de falha:
  - *Omissão* – um processo ou canal deixa de executar as ações que se esperava dele (geralmente devido a pane do processo ou estouro de buffer no canal)
  - *Arbitrária* – um processo ou canal arbitrariamente omite passos de processamento que deveriam ser executados, ou executa passos não intencionados (mais grave tipo de falha, também conhecido como falha bizantina)
  - *Temporização* (apenas para o modelo síncrono) – um processo ou canal não atende os limites de tempo que lhes são estabelecidos (geralmente causada pela sobrecarga dos processos ou da rede)

Classificação das falhas de omissão e arbitrárias

Classe	*Afeta*	*Descrição*
Falha-e-pára	Processo	Processo interrompe sua execução em definitivo. Outros processos podem vir a detectar este estado.
Pane	Processo	Processo interrompe sua execução em definitivo. Outros processos podem não ser capazes de detectar este estado.
Omissão	Canal	Uma mensagem inserida no *buffer* de saída do remetente nunca chega ao *buffer* de entrada do destinatário.
Omissão-envio	Processo	Um processo completa um *envio*, mas a mensagem não é inserida no seu *buffer* de saída.
Omissão-receb.	Processo	Uma mensagem é inserida no *buffer* de entrada de um processo, mas o processo não a recebe.
Arbitrária (Bizantina)	Processo ou canal	Processo/canal exibe um comportamento arbitrário: envio ou recebimento de mensagens arbitrárias em momentos arbitrários; omissões; interrupção ou ação incorreta de um processo.

Classificação das falhas de temporização

Classe	*Afeta*	*Descrição*
Relógio	Processo	Relógio local do processo excede os limites de sua taxa de desvio do tempo real.
Desempenho	Processo	Processo excede os limites do intervalo esperado entre dois passos.
Desempenho	Canal	A transmissão de uma mensagem atrasa além do limite estabelecido.

Problema dos generais bizantinos – Versão 2

- Contexto do problema:
  - Suponha que os bárbaros estejam em número suficiente para derrotar qualquer uma das duas divisões, se atacadas separadamente
  - Enquanto não são atacadas, as duas divisões enviam mensagens regularmente uma à a outra, para reportar a sua situação
- Questão:
  - Como saber se a outra divisão continua acampada, ou já teria sido derrotada?
- Considerações sobre as soluções:
  - No modelo assíncrono, nenhuma divisão consegue distinguir se a outra foi derrotada ou não (por quê?)
  - No modelo síncrono, a derrota de uma divisão pode ser determinada com exatidão (como?), apesar de não ser possível determinar precisamente há quanto tempo o massacre teria acontecido (por quê?)

Problema dos generais bizantinos – Versão 3

- Contexto do problema:
  - Suponha agora que os bárbaros eventualmente possam capturar qualquer um dos mensageiros, impedindo, assim, que algumas mensagens trocadas entre as duas divisões cheguem a seu destino
- Questão:
  - Como estabelecer um acordo entre as divisões, de modo a decidir se (e quando) elas devem atacar o inimigo, ou, no caso de uma das divisões ter sido derrotada, se a outra divisão deve se render
- Considerações sobre as soluções:
  - O estabelecimento de um acordo (ou consenso) entre as duas divisões nessas condições é impossível nos dois modelos de interação!! (prova?)
  - No problema original, os mensageiros ainda podem ser corrompidos para alterar o conteúdo das mensagens (falhas arbitrárias)

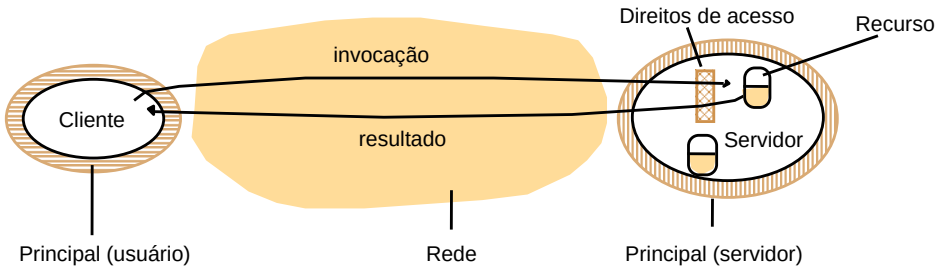
Modelo de segurança

- Descreve os mecanismos utilizados para garantir a segurança dos processos e de seus canais de comunicação, e para proteger os recursos que os processos encapsulam contra acessos não autorizados
- Principais questões:
  - Proteção dos recursos
  - Segurança dos processos e de suas interações
- Desafios:
  - Uso de técnicas de segurança implica em custos substanciais de processamento e de gerência
  - Necessidade de uma análise cuidadosa das possíveis fontes de ameaças, incluindo ambientes externos ao sistema (rede, físico, humano, etc)

Proteção de recursos

- Conceitos envolvidos:
  - *Direitos de acesso* – especificação de quem pode realizar as operações disponíveis para um determinado recurso
  - *Principal* – entidade (usuário ou processo) autorizada para solicitar uma operação, ou para enviar os resultados de uma operação para o principal solicitante
- Responsabilidade compartilhada entre clientes e servidores
  - Servidor verifica a identidade do principal por trás de cada invocação e checa se ele tem direitos de acesso suficientes para realizar a operação solicitada
  - Cliente verifica a identidade do principal por trás do servidor para garantir que os resultados vêm do servidor requisitado

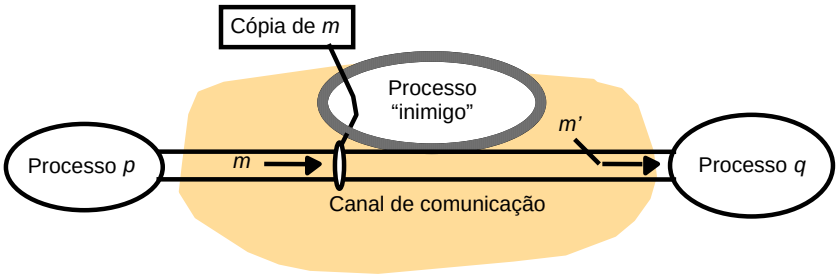
Proteção de recursos



Segurança dos processos e suas interações

- Natureza aberta da rede e dos serviços de comunicação expõe os processos a ameaças e ataques “inimigos”
- Principais tipos de ameaça:
  - *Aos processos* – cliente e servidores podem não ser capazes de determinar a identidade dos processos com os quais se comunicam
  - *Aos canais de comunicação* – mensagens podem ser indevidamente copiadas, alteradas, forjadas ou removidas enquanto transitam pela rede
  - *Negação de serviço* – envios excessivos de mensagens ou invocações de serviços através da rede, resultando na sobrecarga dos recursos físicos do sistema e prejudicando seus usuários
  - *Mobilidade de código* – ameaças disfarçadas na forma de código móvel que deve ser executado localmente pelos clientes (“Cavalo de Tróia”)

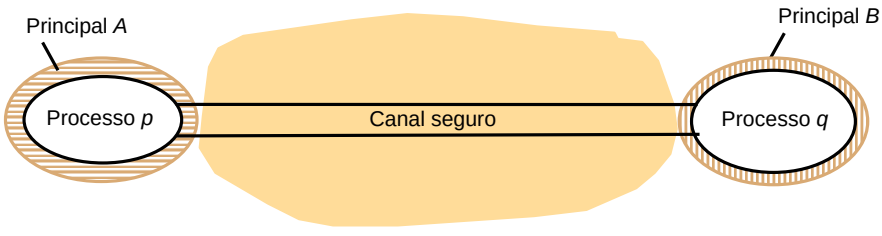
Segurança dos processos e suas interações



Comunicação segura

- Principais mecanismos:
  - Criptografia – processo de “embaralhamento” de uma mensagem de modo a esconder seu conteúdo de usuários não autorizados
  - Autenticação – utiliza chaves secretas e criptografia para garantir a identidade dos processos clientes e servidores
  - Canal seguro – canal de comunicação conectando um par de processos, onde cada processo conhece e confia na identidade do principal em nome do qual o outro processo está executando
    - Geralmente implementado como uma camada extra de serviço sobre os serviços de comunicação existentes
    - Utiliza mecanismos de autenticação e criptografia para garantir a privacidade e a integridade das mensagens transmitidas através do canal
    - Também pode garantir a entrega e ordem de envio das mensagens
    - Ex.: VPN, SSL

Comunicação segura



Exercícios

- No livro: 2.1–2.8, 2.11–2.15, 2.17, 2.18