Sistemas Distribuídos

Rui Oliveira
rco@di.uminho.pt

Programa

- Introdução aos sistemas distribuídos
- Arquitectura de sistemas distribuídos
- Coordenação distribuída

Bibliografia

- Coulouris, Dollimore, Kindberg: Distributed Systems: Concepts and Design (Addison-Wesley) 1994
- Singhal, Shivaratri: Advanced Concepts in Operating Systems (McGraw Hill) 1994
- Silberschatz, Galvin, Gagne: Applied Operating System Concepts (Wiley) 2000

Programa detalhado

- Introdução aos sistemas distribuídos
  - Características e motivações
  - Aspectos de concepção
  - Espectativas de utilização

Introdução aos sistemas distribuídos

- Sistema Distribuído: conjunto de computadores ligados em rede, com software que permite a partilha de recursos e a coordenação de actividades, oferecendo idealmente um ambiente integrado.

  [Coulouris et al.]

Introdução aos sistemas distribuídos

- Um sistema distribuído é um sistema em que a falha de uma máquina da qual nunca se ouviu falar pode pode tornar a nossa própria máquina inutil.

  [Leslie Lamport]

História

- Inicialmente sistemas centralizados multi-utilizador.
- SD originados com o aparecimento de LAN's (1970+) interligando estações de trabalho.
- Fomentados por PC's de baixo custo e alto desempenho.
- Evolução desde LAN's até à WAN global que é a Internet.

SD: Generalidades

- Entidades de processamento independentes que comunicam por troca de mensagens.
- Possibilidade de falhas independentes.
- Heterogeneidade de hardware e software.
- Nos sistemas assíncronos não existe:
  - limite para as velocidades relativas de processamento,
  - limite para o tempo de comunicação,
  - relógio global.

Motivações

- Unix distribuído
  - Fornecer um modelo do UNIX com melhores performances e facilidades do que em computadores multi-utilizador.
  - Computadores com poder de processamento para cada utilizador e servidores para acesso a recursos partilhados (eg. impressoras, disco).

Motivações

- WANs
  - Crescimento exponencial da Internet.
  - Software (tem de ser) projectado tendo em conta a escalabilidade.
  - Resolução de nomes, encaminhamento, correio electrónico, news, WWW, etc.

Motivações

- Aplicações comerciais:
  - Aplicações "sérias" como:
    - Reserva de bilhetes em companhias de aviação,
    - Sistema Multibanco,
    - etc.
  - Exigem fiabilidade e segurança mesmo em presença de falhas de computadores e canais de comunicação.

Motivações

- Aplicações interactivas e multimédia:
  - Dois aspectos ortogonais: largura de banda e latência.
  - Aplicações interactivas, mesmo com poucas necessidades em largura de banda, podem exigir baixa latência.
  - Transmission Control Protocol versus User Datagram Protocol.
  - Jogos multi-utilizador usam tipicamente UDP para comunicação.

Características desejáveis de um SD

- Partilha de recursos
- Abertura
- Concorrência
- Escalabilidade
- Tolerância a faltas
- Transparência
- Estas características não são consequência directa da distribuição. Pelo contrário, o sistema e aplicações deverão ser concebidos de forma a garantir estas características.

Partilha de recursos

- Hardware: impressora, discos, scanner, etc.
- Software: ficheiros, bases de dados, compiladores, etc.
- A partilha reduz custos
- Necessária como suporte ao trabalho cooperativo.

Partilha de recursos

- Encapsulamento dos recursos pela pela máquina que os contém.
- Gestores de recursos de cada tipo: oferecem interfaces standard para aceder aos recursos.
- Motivam modelo cliente/servidor / modelo baseado em objectos.

Abertura

- Extensibilidade do software e hardware com possível heterogeneidade.
- Obtida especificando e documentando interfaces.
- O UNIX é exemplo de um sistema concebido com a Abertura em mente.
- Um sistema distribuído aberto possui um mecanismo uniforme de comunicação entre processos.

Concorrência

- Existe e deve ser suportada em SD's por vários motivos:
  - Vários utilizadores/aplicações podem invocar comandos simultaneamente.
  - Um servidor deve poder responder concorrentemente a vários pedidos que lhe cheguem.
  - Vários servidores devem poder correr concorrentemente colaborando na resolução de pedidos.

Escalabilidade

- O software de ser pensado de modo a funcionar eficaz e eficientemente em sistemas de escalas diferentes.
- Tanto o sistema como as aplicações não deverão necessitar de alterações com o aumento da escala.
- O trabalho envolvido no processamento de acessos a um recurso partilhado deverá ser independente do tamanho do sistema.

Escalabilidade

- Devem ser evitados algoritmos e estruturas de dados centralizadas.
- Nenhum recurso deve ser escasso -em caso de necessidade deve ser possivel aumentar o número de recursos de forma a satisfazer a procura.
- Soluções passam por replicação de dados, caching, algoritmos distribuídos.

Tolerância a faltas

- Os sistemas por vezes falham. Na presença de falhas, os programas podem produzir resultados errados ou terminar prematuramente.
- Soluções passam por redundância de hardware e/ou de software (servidores/serviços).
- Quando um componente falha, apenas o trabalho que usa este componente deve ser afectado e deve poder re-iniciar ou continuar usando outro componente.

Tolerância a faltas

- Em SD a disponibilidade pode ser maior que em sistemas centralizados mas exige maior complexidade do software.
- Motiva paradigmas mais avançados que cliente servidor (eg. comunicação em grupo, transacções).

Transparência

- O sistema deve ser visto como um todo e não como uma coleccção de componentes distribuídos.
- Tipos de transparência: acesso, localização, concorrência, replicação, falhas, migração, desempenho e escalabilidade.
- Transparência de acesso + localização = transparência de rede.
- Exemplo: email. Contra-exemplo: rlogin.

Programa detalhado

- Introdução aos sistemas distribuídos
  - Características e motivações
  - Aspectos de concepção
  - Espectativas de utilização

Aspectos da concepção de um SD

- Nomes
- Comunicação
- Estrutura do software
- Alocação de carga
- Coêrencia
- Estes aspectos de concepção são consequência directa da distribuição. Outros, mais genéricos, continuam a ser importantes.

Nomes

- Os SD são baseados na partilha de recursos e na transparência da sua distribuição.
- O significado deve ser global e independente da localização do recurso.
- Necessidade de resolução de nomes (eg. hostname → IP).
- Mecanismo de tradução deve ser escalável e eficiente.

Nomes

- Espaço de Nomes pode ser simples (flat) ou estruturado (hierárquico).
- Os nomes são resolvidos dentro de Contextos.
- Serviço de Nomes:
  - Traduz Nomes de um Espaço de Nomes para outro.
  - Um Nome é resolvido quando é obtido um identificador que permite a invocação de uma acção sobre o recurso a que diz respeito o Nome.

Comunicação

- Os processos de um SD encontram-se logica e fisicamente separados interagindo por troca de mensagens.
- A comunicação entre dois processos envolve:
  - a transferência de dados do espaço de endereçamendo do emissor para o do receptor.
  - por vezes é necessária a sincronização do receptor com o emissor, de forma que o emissor ou o receptor é bloqueado até o outro o permitir libertar.

Comunicação

- Número e complexidade das camadas de software.
- Compromisso entre desempenho e modelo de programação de alto nível.
- Necessidades: transferência de dados e sincronização.

Comunicação

- Primitivas de comunicação:
  - send e receive realizam a passagem de mensagens entre pares de processos.
  - envolve a transmissão de dados (mensagem) através dum meio de comunicação especificado (canal) e a recepção da mensagem.
  - a transmissão pode ser:
    - assíncrona: a mensagem é enviada e o emissor prossegue.
    - síncrona: o emissor espera até a mensagem ser recebida.

Comunicação

- Padrões de comunicação:
  - Send-Receive básico
  - Cliente-Servidor
    - Prestação de serviços
    - pedido-execução-resposta
  - Difusão
    - destinatário é um grupo de processos
    - a um send corresponde um receive por membro do grupo

Comunicação

- Diferentes modelos de programação:
  - Primitivas básicas tipo send/receive (eg. sockets).
  - Invocação remota de procedimentos (eg. RPC).
  - Invocação de operações em sistemas de objectos distribuídos (eg. Java RMI, CORBA).
  - Padrões de comunicação de mais alto nível assegurando ordem, atomicidade, domínios de filiação.

Estrutura do software

- Num sistema centralizado o SO é tipicamente monolítico e oferece uma interface fixa.
- Em sistemas distribuídos o kernel pode assumir um papel mais restrito gerindo os recursos mais básicos.
- É dado ênfase a serviços, que podem ser fornecidos por processos em modo utilizador.

Estrutura do software

- Categorias de software num SD:
  - Aplicações
  - Sistema operativo
    - gestão básica de recursos:
      - alocação e protecção de memória
      - criação e escalonamento de processos
      - comunicação entre processos
      - gestão de periféricos
    - abstracções normais de programação
    - garantias de confiabilidade e segurança do SO

Estrutura do software

- Categorias de software num SD:
  - Serviços
    - Funcionalidades comuns necessárias à programação de aplicações distribuídas
    - Oferece modularidade e adaptabilidade aos sistema
    - Serviços básicos (ficheiros) até de alto nível (email)
  - Suporte à programação
    - Funcionalidades em runtime de suporte a primitivas / construções oferecidas pelas linguagens/ambientes de programação.

Alocação de carga

- Modelos:
  - Workstation-Server
  - Processor pool
  - Uso de workstations desocupadas

Alocação de carga

- Workstation-Server.
  - Alocação da carga nas estações de trabalho (perto do utilizador) -previligia aplicações interactivas.
  - Capacidade da estação de trabalho determina o tamanho da maior tarefa do utilizador.
  - Não optimiza o uso das capacidades (processamento e memória) através do SD.
  - Um utilizador não consegue obter mais recursos dos que os locais.

Alocação de carga

- Processor pool (eg. Amoeba)
  - Alocação dinâmica de processadores a utilizadores num modelo baseado no Workstation-Server.
  - O parque de processadores consiste numa colecção (heterogénia) de computadores de baixo custo.
  - Cada parque tem uma ligação independente à rede.
  - Os processadores são alocados aos processos até estes terminarem.

Alocação de carga

- Uso de workstations desocupadas.
  - As estações de trabalho com baixa carga ou desocupadas são em determinados períodos alocadas a tarefas de outros utilizadores (remotos).
  - É uma mistura dos modelos anteriores

Coerência

- Coerência de:
  - Actualizações
  - Replicação
  - Cache
  - Falhas
  - Interface com o utilizador

Coerência

- De actualizações:
  - Acontece quando vários processos acedem e actualizam os mesmos dados concorrentemente.
  - A actualização de um conjunto de dados não é executada instantaneamente.
  - A actualização deve, no entanto, ser atómica. Deve parecer instantânea.

Coerência

- De replicação:
  - Se os dados de determinado item forem copiados para vários computadores e posteriormente alterados num ou em vários deles, então surge a possibilidade de inconsistências dados os vários valores para o item nos diversos computadores.
- De cache:
  - Surge sempre que os dados de determinado item remoto são guardados em cache e o valor do item é alterado (similar à replicação).

Coerência

- De falhas:
  - Quando um componente crucial ao sistema falha, os outros inevitavelmente falharão (mesmo que forçados).
  - Dependendo da altura em que cada componente falhe os seus estados podem divergir.
- De interface com o utilizador:
  - Incoerências provocadas pela latência de input/ processamento/output e de comunicação.

Programa detalhado

- Introdução aos sistemas distribuídos
  - Características e motivações
  - Aspectos de concepção
  - Espectativas de utilização

Expectativas de utilização

- Funcionalidade
- Reconfiguração
- Qualidade de serviço

Funcionalidade

- Minímo - a funcionalidade oferecida pelo SD deve ser no minímo a esperada de um único computador.
- Esperado - o SD deve oferecer uma melhoria sobre o serviço de qualquer computador isolado -eg. uma maior variedade de serviços e recursos.

Reconfiguração

- Capacidade de acomodar alterações sem por em causa o sistema.
- Os SD devem poder suportar:
  - alterações rápidas, eg. falha de um processo ou canal de comunicação, variações de carga, migração de recursos ou dados.
  - de médio e longo termo, eg. mudanças de escala, acomodação de novos e diferentes componentes, mudança de função dos componentes.

Qualidade de serviço (QoS)

- Desempenho
  - tempos de resposta
- Fiabilidade
  - desvios da especificação
- Disponibilidade
  - resiliência
- Segurança
  - privacidade e integridade

Programa

- Introdução aos sistemas distribuídos
- Arquitectura de sistemas distribuídos
- Coordenação distribuída

Programa detalhado

- Arquitectura de sistemas distribuídos
  - Generalidades e conceitos
  - Paradigmas e modelos de programação
  - Sistemas operativos distribuídos
  - Comunicação distribuída

Arquitectura de sistemas distribuídos

- O objectivo de uma arquitectura para sistemas distribuídos é a standarização dos componentes do sistema, das formas como interagem e de como são geridos.
- O interesse em arquitecturas para sistemas distribuídos remonta a 1984 com a ANSA - Advanced Network Systems Architecture.
- ...DCE, CORBA, ONC, DCOM...

Interesse de uma arquitectura

- Descrição: forma de descrever a estrutura do sistema distribuído.
- Interoperabilidade: aplicações que usem a mesma arquitectura deverão interagir sem dificuldades.
- Facilita o desenvolvimento: incentiva o desenvolvimento por refinamentos.
- Confiabilidade: a modularidade incentiva o isolamento de falhas.

Aspectos de concepção

- As arquitectura para sistemas/aplicações distribuídas priviligiam os seguintes aspectos:
  - Independência de localização
  - Gestão de conectividade
  - Escalabilidade
  - Performance
  - Securança
  - Equilíbrio de carga
  - Tolerância a faltas

Programa detalhado

- Arquitectura de sistemas distribuídos
  - Generalidades e conceitos
  - Paradigmas e modelos de programação
  - Sistemas operativos distribuídos
  - Comunicação distribuída

Paradigmas

- CPU centered
- Storage centered
- Communication centered

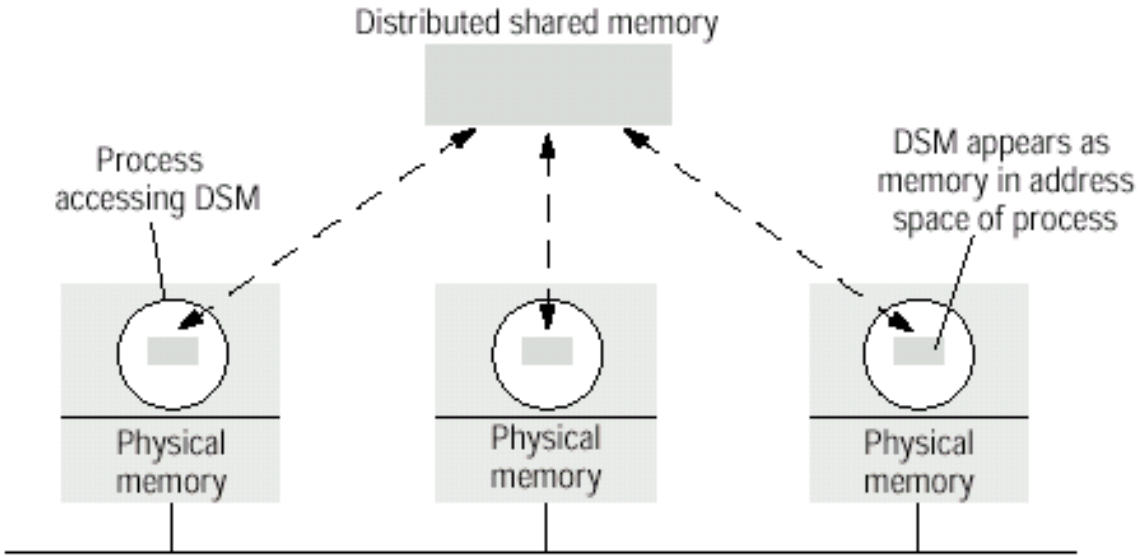
Paradigmas

- Sistema centrado no processador:
  - Data shipping:
    - Sessão remota
    - Transferência explícita de ficheiros.
    - Sistema de ficheiros em rede.
  - Problemas:
    - desempenho
    - controlo de concorrência
    - confiabilidade, etc.

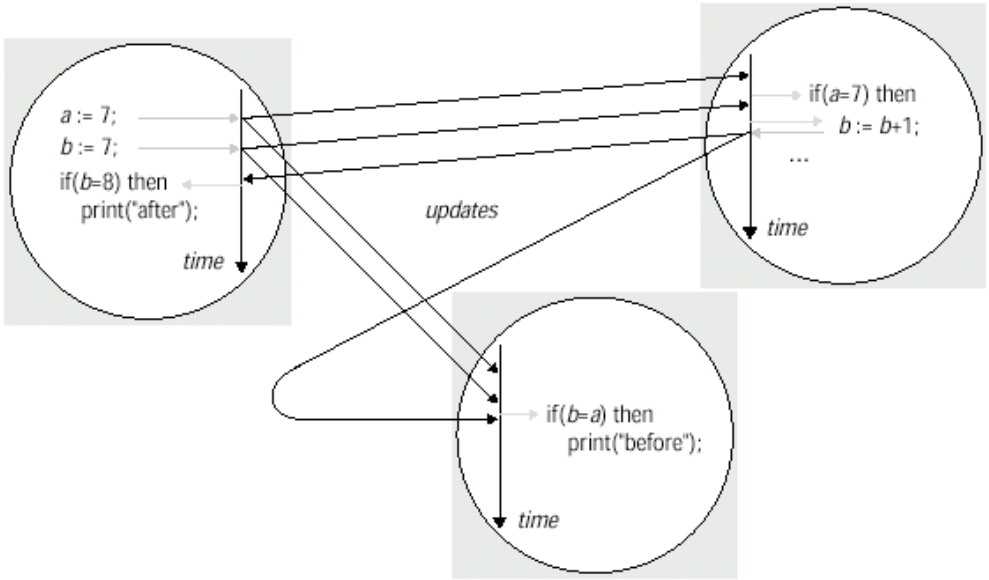
Paradigmas

- Sistema centrado no armazenamento
  - DSM (Distributed Shared Memory)
- \+ Abstracção, adaptação de aplicações
- -Complexidade, modelo de coerência, desempenho
  - Function shipping:
    - Passagem de mensagens
    - Cliente / Servidor

DSM



DSM

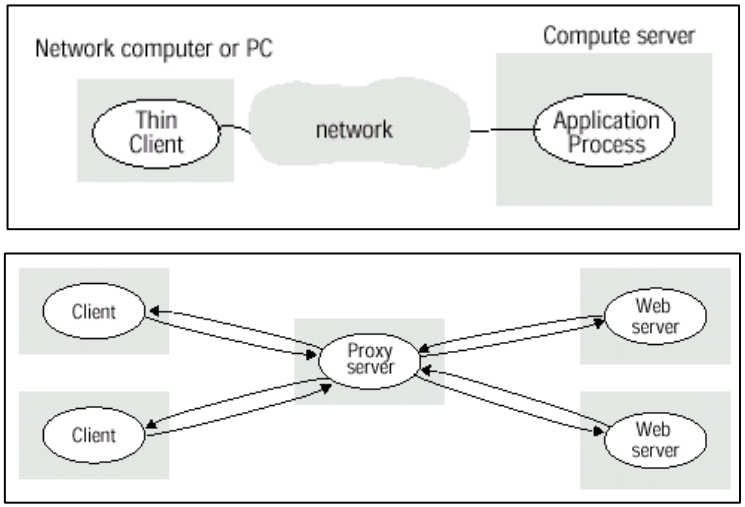


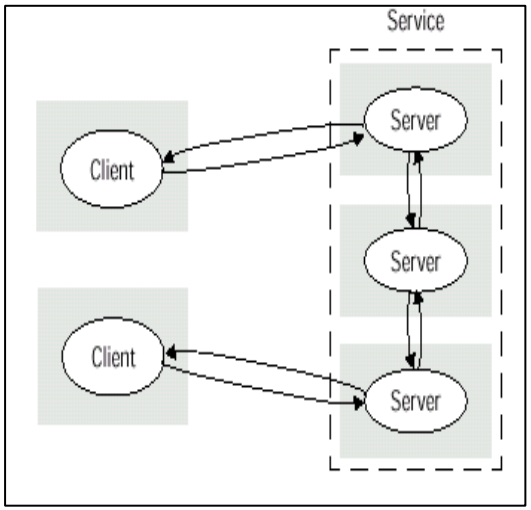
Paradigmas

- Sistema centrado na comunicação
  - Generalização de cliente/servidor
  - Ênfase em serviços
  - Barramento de dados e funções
  - Barramento de objectos - mediador de pedidos

Cliente/Servidor

- A maioria (99%) dos sistemas distribuídos actuais é baseada no paradigma Cliente/Servidor





Cliente/Servidor

- Servidor
  - grande capacidade de armazenamento e/ou processamento.
  - periféricos especiais. Ex. impressão, salvaguarda, ...
  - serviços particulares. Ex. pesquisa, autenticação, ...
- Cliente
  - interface com o utilizador
  - caching

Cliente/Servidor

- Exemplos de servidores vulgares:
  - ficheiros
  - base de dados
  - informação de rede
  - nomes
  - email
  - autenticação

Cliente/Servidor

- Servidor diz-se sem estado (stateless) se:
  - de uma invocação para a seguinte, o servidor não mantém nenhuma informação acerca do estado do cliente (eg. não "sabe" se o cliente guarda algo em cache)
  - o cliente é o único responsável pelas suas acções, devendo portanto confirmar sempre se o que tem em cache ainda é válido
  - exemplo
    - Web proxies, web servers, NFS

Cliente/Servidor

- Servidor diz-se com estado (stateful) se:
  - o servidor mantém informação sobre os clientes com "sessões em aberto" (eg. o servidor é dono da cache e faz chamadas aos clientes quando os dados são alterados
  - o cliente pode confiar na validade da cache
  - exemplo
    - TP monitors, database servers

Cliente/Servidor

- Servidor stateless executa a operação pedida e "esquece" o cliente
  - recupera muito depressa após um crash
  - tipicamente precisa de mais argumentos em cada invocação
  - o "estado" está no cliente e é passado ao servidor, quando necessário (eg. cookies)

Cliente/Servidor

- Servidor stateful sabe "tudo" acerca dos clientes
  - pode ser mais eficiente, pois tem mais informação
  - é complicado recuperar o estado perdido após uma falha: pedir a outro(s) processo(s) que lho passem? Quem são? Clientes? São de confiança? Estão acessíveis? Grupo de servidores replicados? E se foi uma falsa suspeita de falha?
  - estado em memória persistente demora a actualizar e a recuperar

Programa detalhado

- Arquitectura de sistemas distribuídos
  - Generalidades e conceitos
  - Paradigmas e modelos de programação
  - Sistemas operativos distribuídos
  - Comunicação distribuída

Sistemas Operativos Distribuídos

- O objectivo de um SOD é permitir programar um sistema distribuído de forma a que este possa ser usado pelo mais vasto leque de aplicações.
- Este objectivo consegue-se oferecendo às aplicações abstracções, gerais e orientadas ao problema em mãos, dos diversos recursos do sistema. Exemplo disto é oferecer às aplicações processos e canais de comunicação em vez de processadores e uma rede de dados.

Sistemas Operativos Distribuídos

- Um SOD é constituído por um conjunto de núcleos (kernels) e de servidores (processos em modo utilizador).
- Este conjunto de núcleos e de servidores forma uma infraestrutura genérica e transparente em termos de rede de comunicação, para a gestão de recursos.

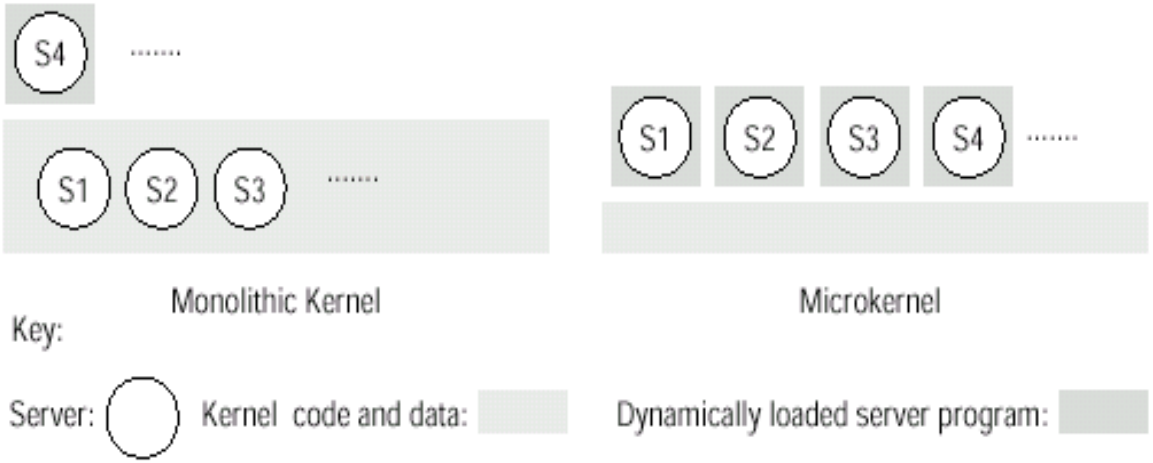
Encapsulamento de recursos

- Um SOD deve tornar os seus recursos acessiveis em toda a rede de comunicação salvaguardando:
  - Encapsulamento: cada recurso deve ter uma interface bem definida para acesso pelos clientes. Todos os detalhes de implementação e gestão do recurso devem ser escondidos.
  - Concorrência: os clientes devem poder partilhar e aceder concorrentemente aos rescursos.
  - Protecção: os recursos devem ser protegidos contra acesso ilegítimos.

Acesso aos recursos

- Os clientes acedem ao recursos identificando-os em argumentos de chamadas aos sistema.
- Ao acesso a um recurso chamamos invocação. A invocação tem associadas tarefas que competem ao SOD:
  - resolução de nomes
  - comunicação
  - escalonamento de acessos

Núcleos monolíticos e micro-núcleos



Núcleos monolíticos

- O UNIX é normalmente considerado monolítico:
  - Executa todas as operações básicas (algumas que não requerem necessariamente privilégios especiais).
  - Consiste num único bloco de código de funcionalidades indiferenciadas.
  - A falta de modularidade faz com que a alteração e adaptação de componenetes individuais seja difícil e se repercuta nos resto do núcleo.

Micro-núcleos

- Um micro-núcleo oferece apenas as funcionalidades mais básicas: gestão de processos, gestão de memória e comunicação entre processos.
- Todos os outros serviços do sistema são providos por servidores activados dinamicamente. Estes serviços são activados nas máquinas das quais se espera o serviço.

Micro-núcleos

- Os serviços do sistema são acedidos através de invocações (normalmente por RPC).

Núcleos monolíticos vs. Micro-núcleos

- Os micro-núcleos destacam-se pela sua abertura e modularidade salvaguardando a protecção de memória.
- Em núcleos monolíticos é muito difícil extrair módulos de forma a poderem executar remotamente noutra máquina.
- A (grande) vantagem dos núcleos monolíticos é o desempenho.

Arquitectura de micro-núcleos

- Os micro-núcleos devem ser transportáveis entre diferentes arquitecturas de computadores.
- Os seus componentes principais incluem:
  - Gestor de processos: criação e gestão a baixo-nível de processos.
  - Gestor de threads: criação, sincronização e escalonamento.
  - Gestor de memória: gestão da memória fisíca.
  - Supervisor: processamento de interrupções e chamadas ao sistema.

Exemplos de micro-núcleos

- Mach
- Chorus
- Amoeba
- Clouds

Programa detalhado

- Arquitectura de sistemas distribuídos
  - Generalidades e conceitos
  - Paradigmas e modelos de programação
  - Sistemas operativos distribuídos
  - Comunicação distribuída

Comunicação distribuída

- A abstracção básica da comunicação distribuída é a passagem de mensagens.
- Para a maior parte dos sistemas/aplicações o uso directo de primitivas send / receive revela-se de demasiado baixo nível.
- A abstracção para comunicação distribuída por excelência é a invocação remota de procedimentos -RPC.

Mapeamento da estrutura de dados

- Os dados num programa são representados por estruturas mais ou menos complexas. Os dados nas mensagens são sequenciais, em cadeia.
- As estruturas de dados têm que ser "achatadas" no envio e reconstruidas na recepção.
- Os dados numa mensagem podem ser de diferentes tipos.

Mapeamento da estrutura de dados

- Nem todos os computadores representam os valores (inteiros, caracteres) da mesma forma.
- A troca de dados entre dois computadores:
  - requer que os valores sejam convertidos antes do envio para um formato externo pré-acordado e covertidos para o formato local na recepção.

Mapeamento da estrutura de dados

- A troca de dados entre dois computadores (cont.):
  - ou o envio dos dados na sua forma original acompanhados pelo tipo de arquitectura do emissor. O receptor deverá ser capaz de fazer a transformação necessária.
  - não requer a conversão entre computadores do mesmo tipo.

Primitivas de envio e recepção

- A passagem de mensagens é suportada por duas primitivas: enviar e receber.
- Um processo envia uma mensagem para o destinatário e outro processo, no destino, recebe a mensagem.
- A cada destinatário é associada uma fila: o emissor coloca mensagens na fila e o destinatário remove-as.

Comunicação síncrona

- O emissor e receptor sincronizam em cada mensagem.
- O envio e a recepção são operações bloqueantes.
- Aquando do envio, o emissor bloqueia até à recepção da mensagem.
- Aquando da recepção, o receptor bloqueia até chegar uma mensagem.

Comunicação assíncrona

- O envio não é bloqueante: o emissor pode continuar a sua execução logo após o envio da mensagem (que é copiada para um buffer local).
- A transmissão da mensagem processa-se em paralelo com a execução do emissor.
- A recepção pode ou não ser bloqueante.

Identificação do destinatário

- O envio de uma mensagem requer a indicação do identificador do(s) seu(s) destinatário(s).
- Identificadores independentes da localização:
  - Quando o identificador do destinatário é independente da localização o identificador é, aquando o envio, mapeado para um endereço de baixo nível onde de facto é entregue a mensagem.
  - O uso de identificadores independentes da localização permite a recolocação de serviços sem aviso prévio aos clientes.

Destinatários das mensagens

- Processos
- Portos
- Sockets
- Grupos de processos
- Grupos de portos
- Objectos

Fiabilidade

- Entrega não fiável:
  - a mensagem é transmitida uma única vez sem espera de confirmação (ack).
  - não há garantia de que a mensagem enviada seja de facto entregue (eg. UDP).
  - ao utilizar este serviço, fica a cargo dos processos assegurar a fiabilidade de que precisam.

Fiabilidade

- Entrega fiável:
  - Pode ser implementada sobre a entrega não fiável através de confirmações de recepção.
  - Cada mensagem deverá ter um identificador único.

Comunicação Cliente/Servidor

- Suporte ao paradigma cliente/servidor
- Primitivas:
  - DoOperation: usado pelos clientes para a invocação de operações remotas.
  - GetRequest: usado pelos servidors para obter pedidos de serviço.
  - SendReply: usado pelos servidores para enviar respostas aos clientes.

Comunicação Cliente/Servidor

- A resposta do servidor funciona como confirmação da invocação.
- Protocolos de invocação remota de procedimentos (RPC):
  - R: Request
    - Pode ser usado quando o procedimento não devolve nada e o cliente não pretende confirmação da recepção.
    - O cliente não bloqueia.

Comunicação Cliente/Servidor

- Protocolos de invocação remota de procedimentos (cont.):
  - RR: Request-Reply
    - Comunicação cliente/servidor típica.
  - RRA: Request-Reply-Acknowledgement
    - O cliente confirma a resposta do servidor.

Comunicação em grupo

- Na comunicação entre um grupo de processos é utilizada a difusão de mensagens.
- A difusão de mensagem é utilizada na implementação de sistemas distribuídos com as seguintes características:

Comunicação em grupo

- Tolerância a faltas baseada em serviços replicados
  - um serviço replicado consiste num conjunto de servidores. As invocações dos clientes são difundidas a todos os processos do grupo que executam a mesma operação.
- Localização de processos num sistema distribuído
- Melhor desempenho através de dados replicados
- Notificações múltiplas

Comunicação em grupo

- Propriedades dos protocolos de comunicação em grupo:
  - Difusão fiável
    - Todos os processos que não falhem recebem a mensagem.
  - Difusão atómica
    - Difusão fiável com ordem total na entrega das mensagens.

Programa

- Introdução aos sistemas distribuídos
- Arquitectura de sistemas distribuídos
- Coordenação distribuída

Programa detalhado

- Coordenação distribuída
  - Ordenação
  - Exclusão mútua
  - Atomicidade
  - Deadlocks
  - Acordo e Eleição

Ordenação de eventos

- Causalidade
- Relação Happened-before (→)
  - Se A e B são eventos do mesmo processo e A foi executado antes de B, então A→B.
  - Se A é o evento de envio de uma mensagem por um processo e B o evento de recepção dessa mensagem por outro processo, então A→B.
  - Se A→B e B→C então A→C

Realização de →

- Associa-se um etiqueta temporal a cada evento do sistema de forma a que se A→B então a etiqueta de A é menor que a etiqueta de B.
  - Cada processo tem associado um relógio lógico. O relógio é um contador que é incrementado entre cada dois eventos sucessivos do processo.
  - Cada mensagem enviada transporta o instante lógico em que foi enviada.
  - Ao receber uma mensagem, o processo acerta o seu relógio com o instante da mensagem se este último for mais recente.

Programa detalhado

- Coordenação distribuída
  - Ordenação
  - Exclusão mútua
  - Atomicidade
  - Deadlocks
  - Acordo e Eleição

Exclusão mútua distribuída

- Assumpções
  - O sistema consiste de n processos; cada processo no seu processador.
  - Cada processo tem uma zona crítica que requer exclusão mútua.
- Requisitos
  - Se um processo se encontra a executar a sua zona crítica, então mais nenhum processo se encontra a executar a sua.

Exclusão mútua distribuída: alg centralizado

- Um processo é escolhido para coordenar o acesso à zona crítica.
- Um processo que queira executar a sua zona crítica envia um pedido ao coordenador.
- O coordenador decide que processo pode entrar na zona crítica e envia a esse processo uma resposta.
- Quando recebe a resposta do coordenador, o processo inicia a execução da sua zona crítica.
- Quando termina a execução da sua zona crítica, o processo envia uma mensagem a libertar a zona

Exclusão mútua distribuída: alg distribuído

- Quando um processo Pi pretende aceder à sua zona crítica gera uma etiqueta temporal TS e envia um pedido (Pi, Tsi) a todos os processos.
- Quando um processo recebe um pedido pode responder logo ou adiar a sua resposta.
- Quando um processo recebe respostas de todos os processos no sistema pode então executar a sua zona crítica.
- Depois de terminar a execução da sua zona crítica, o processo responde a todos os pedidos aos quais adiou a resposta.

Exclusão mútua distribuída: alg distribuído

- A decisão de Pj responder logo a um pedido (Pi, TSi) ou adiar depende de três factores:
  - Se Pj estiver na sua zona crítica, adia.
  - Se Pj não pretender aceder à sua zona crítica, responde.
  - Se Pj pretender aceder à sua zona crítica (e já enviou também um pedido) então compara a etiqueta temporal do seu pedido TSj com TSi:
    - Se TSj > TSi então responde logo (Pi pediu primeiro)
    - Senão adia a resposta.

Exclusão mútua distribuída: alg distribuído

- Comportamento desejado da solução distribuída:
  - Não há deadlocks.
  - Não há starvation. Garantido pela ordem total dada pelos relógios lógicos.

Exclusão mútua distribuída: alg distribuído

- Problemas da solução distribuída:
  - Cada processo tem de conhecer todos os outros. Conjunto estático.
  - Não tolera a falha de nenhum processo.

Programa detalhado

- Coordenação distribuída
  - Ordenação
  - Exclusão mútua
  - Atomicidade
  - Deadlocks
  - Acordo e Eleição

Atomicidade

- Dada uma sequencia distribuída de operações, ou são todas executadas ou não é nenhuma - a esta sequencia chamamos transacção.
- A atomicidade num sistema distribuído requer um coordenador de transacções responsável por:
  - Iniciar a transacção.
  - Distribuir a transacção pelos respectivos nodos.
  - Coordenar a terminação da transacção que resulta no sucesso (commit) ou falha (abort) por todos os nodos.

Compromisso em duas fases

- Protocolo de terminação de transacções (Two-phase commit)
- O protocolo é iniciado após o fim da última operação da transacção.
- O protocolo envolve todos os nodos em que se executa a transacção.

Compromisso em duas fases: fase 1

- Seja T uma transacção e Ci o coordenador:
  - Ci adiciona <preparar T> ao log
  - Ci envia <preparar T> a todos os nodos
  - Quando recebe <preparar T> cada nodo responde consoante o sucesso das operações por si executadas:
    - Se falhou adiciona <no T> ao log e responde a Ci com <abort T>
    - Se teve sucesso adiciona <yes T> ao log, grava o log em disco, e responde a Ci com <commit T>

Compromisso em duas fases: fase 1

- O coordenador Ci recebe as respostas:
  - Se todos os nodos respondem <yes T> então a decisão é commit.
  - Basta que um nodo responda <no T> e a decisão é abort.
  - Basta que um nodo demore a responder (time out) e a decisão é abort.

Compromisso em duas fases: fase 2

- O coordenador Ci :
  - Torna a decisão persistente escrevendo-a em disco.
  - Envia a a decisão a todos os nodos.
- A decisão é irrevogável.
- Todos os nodos actuam de acordo com a decisão.

Programa detalhado

- Coordenação distribuída
  - Ordenação
  - Exclusão mútua
  - Atomicidade
  - Deadlocks
  - Acordo e Eleição

Deadlocks: prevenção

- Controlo de recursos centralizado
  - Um processo é incumbido de ceder controlo a todos os recursos.
- Ordenação de recursos
  - Requer uma ordem total entre os recursos: cada recurso é numerado univocamente
  - Um processo só pode requisitar um recurso se não tiver controlo sobre um recurso com valor maior

Deadlocks: prevenção

- Processos com prioridades e preempção
  - A cada processo é atribuída uma prioridade. Um processo mais prioritário ganha o controlo do recurso ao menos prioritário.
- Processos com idade
  - Cada processo tem uma data de nascimento
  - Wait-die: não preemptivo prioridade aos mais novos
  - Would-wait: preemptivo prioridade aos mais velhos

Deadlocks: detecção

- Algoritmo centralizado
  - Cada nodo mantem um grafo de espera local. Os vertices do grafo correspondem aos processos que detêm ou pretendem recursos do nodo.
  - Um processo coordenador solicita periodicamente os grafos de espera locais e constroi um global com vertices para todos os processos no sistema.
  - Qualquer ciclo no grafo global é um deadlock.

Deadlocks: detecção

- Algoritmo distribuído
  - Cada nodo mantem um grafo de espera local. Os grafos locais têm um vertice adicional Pex extra.
  - Se o grafo local tiver um ciclo que não envolva Pex então há um deadlock.
  - Se o grafo local tiver um ciclo que envolva Pex então há provavelmente um deadlock. Inicia-se um algoritmo de detecção distribuído.

Programa detalhado

- Coordenação distribuída
  - Ordenação
  - Exclusão mútua
  - Atomicidade
  - Deadlocks
  - Acordo e Eleição

Eleição

- Determinar quando e onde um novo coordenador é necessário.
- Assuma-se que cada processo tem uma prioridade única e distinta.
- O coordenador é sempre o processo com mais alta prioridade.

Eleição

- Algoritmo Bully
  - O algoritmo é despoletado por um processo Pi que julga que o coordenador falhou.
  - Pi envia uma mensagem de eleição a todos os processos com maior prioridade que Pi.
  - Se num intervalo T Pi não receber qq resposta, então auto-elege-se coordenador.
  - Se receber alguma resposta então Pi espera durante T' por uma mensagem do novo coordenador.
  - Se não receber nenhuma mensagem então reinicia o algoritmo.

Eleição

- Algoritmo Bully
  - Se Pi não é o coordenador então, a qualquer momento pode receber uma mensagem...

Eleição

- Algoritmo em anel
  - Aplicável a sistemas organizados fisica or logicamente em anel
  - Assume que os canais de comunicação são unidireccionais
  - Cada processo mantem uma lista de activos que consiste nas prioridades de todos os processos activos quando este algoritmo terminar.
  - Quando um processo Pi suspeita a falha do coordenador, Pi cria uma lista de activos vazia, envia uma mensagem de eleição m(i) ao seu vizinho e insere i na lista de activos.

Eleição

- Algoritmo em anel
  - Se Pi recebe uma mensagem de eleição m(j) responde de uma de três formas:
    - Se foi a primeira mensagem de eleição que viu, então cria uma lista de activos com i e j. Envia as mensagens de eleição m(i) e m(j) (nesta ordem) ao vizinho.
    - Se i≠j então junta j à sua lista de activos e reenvia a mensagem ao vizinho.
    - Se i=j então a sua lista de activos já contem todos os processos activos no sistema e Pi pode determinar o coordenador.