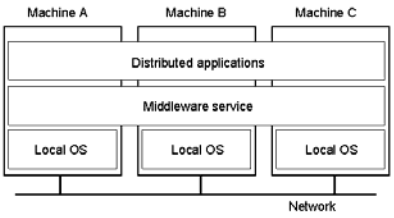
Introdução

Definição de um Sistema Distribuído (1)

Um sistema distribuído é:

Uma coleção de computadores independentes que aparecem para o usuário como um único sistema coerente.

Definição de um Sistema Distribuído(2)



Um sistema distribuído organizado como middleware.
Note que o nível de middleware se extende através de multiplas maquinas.

Metas de Sistemas Distribuídos (2)

- Um sistema distribuído tem como meta principal facilitar o acesso do usuário a recursos remotos e compartilhar estes recursos com outros usuários de uma forma controlada (**Conectar recursos e usuários**).
- Meta importante de um SD é esconder o fato de que seus processos e recursos estão fisicamente distribuídos através de vários computadores (**Transparência**).
- Outra meta importante em SD é **Abertura**. Um SD aberto é um sistema que oferece serviços de acordo com regras padrões.
- Meta de projeto importante em SD diz respeito a capacidade de crescimento do sistema (**Escalabilidade**).

Transparência em Sistemas Distribuídos

Transparência	Descrição
Acesso	Esconde diferenças na representação de dados e como um recurso é acessado
Localização	Esconde onde um recurso está localizado
Migração	Esconde que um recurso pode mover-se para outra localização
Relocação	Esconde que um recurso pode ser movido para outra localização enquanto esta sendo usado
Replicação	Esconde que um recurso pode ser compartilhado por vários usuários concorrentes
Concorrência	Esconde que um recurso pode ser compartilhado por vários usuários concorrentes
Falha	Esconde a falha e recuperação de um recurso
Persistência	Esconde quando um recurso (software) esta em memória ou em disco

Diferentes formas de transparência em um sistema distribuído.

Abertura em Sistemas Distribuídos

- **Sistema Distribuído Aberto**: Capacitado a interagir com serviços de outros sistemas abertos:
  - Sistemas deveriam concordar com interfaces bem definidas
  - Sistemas deveriam suportar portabilidade de aplicações
  - Sistemas deveriam interoperar com facilidade
- **Conseguir Abertura**: No mínimo fazer o SD independente da heterogeneidade do ambiente
  - Hardware
  - Plataformas
  - Linguagens

Política X Mecanismo

• **Implementando abertura**: necessita suporte para diferentes políticas especificadas por aplicações e usuários:
  • que nível de consistência é preciso para dados no cliente?
  • Que operações é permitido código *baixado* realizar?
  • Que requisitos de QoS ajustamos em banda variável?
  • Que nível de segredo necessitamos para comunicação?

• **Implementando abertura** : Ideal, um SD fornece apenas mecanismos:
  • permite ajustar políticas de cache
  •suporta diferentes níveis de confiança em código móvel
  •fornece ajuste de parâmetros de QoS
  •oferece diferentes algoritmos de criptografia

Escala em Sistemas Distribuídos

• **Observação**: muitos desenvolvedores de SDs utilizam facilmente a palavra escalável sem deixar claro porque seus sistemas realmente escalam.

• **Escalabilidade**: mínimo 3 componentes
  • numero de usuários e/ou processos (tamanho)
  • distância máxima entre nodos (geográfica)
  • numero de domínios administrativos (administrativa)

Grande parte dos sistemas contam apenas, escalabilidade em tamanho. Solução: servidores poderosos.

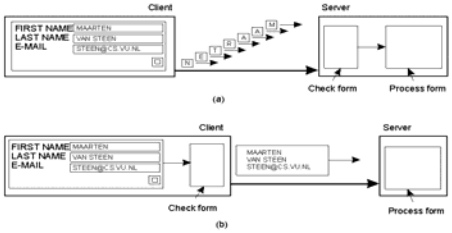
Hoje, desafio é escalabilidade geográfica e administrativa.

Problemas de Escalabilidade

Conceito	Exemplo
Serviços centralizados	Um único servidor para todos os usuários
Dados centralizados	Um único guia telefônico on-line
Algoritmos centralizados	Fazer roteamento basedo em informação completa

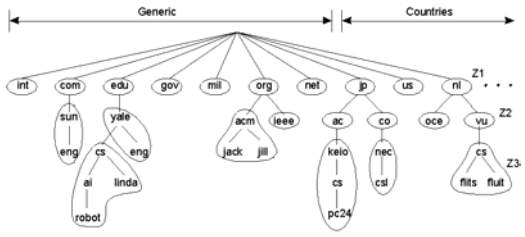
Exemplos das limitações de escalabilidade.

Técnicas de escalabilidade (1)



- Distribuição : dividir dados e computações nas máquinas
- Replicação : fazer cópias disponíveis em várias máquinas
- Caching : permitir processos clientes acessar cópias locais

Técnicas de escala(2)

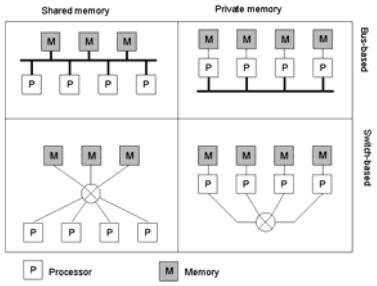


Um exemplo de divisão espaço de nomes DNS em zonas.

Conceitos de Hardware

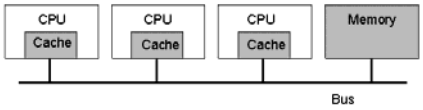
• Diferentes organizações do hardware podem ser consideradas para SDs especialmente com relação a forma de interconexão e comunicação.
  • Multiprocessadores
  • Multicomputadores
  • Redes de Computadores

Multiprocessadores e Multicomputadores



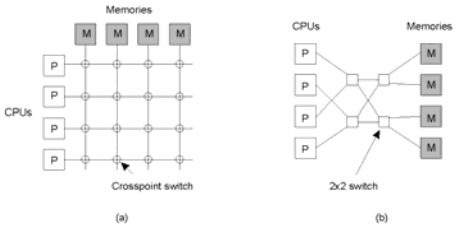
Diferentes organizações basicas: memória compartilhada X privada e interconexão barramento X chaveamento

Multiprocessadores (1)



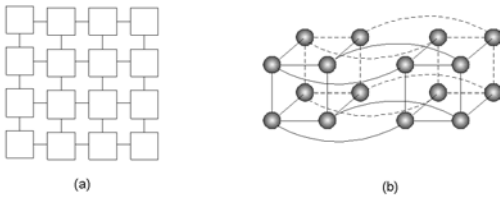
Um multiprocessor baseado em barramento.

Multiprocessadores (2)



a) crossbar
b) omega

Multicomputador Homogêneo



a) Grelha (Grid)
b) Hipercubo

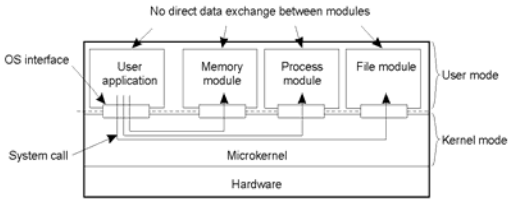
Conceitos de Software

Sistema	Descrição	meta
SOD	Sistemas operacionais fortemente acoplados para multi-processadores e multicomputadores homogêneos	Esconder e gerenciar recursos de hardware
SOR	Sistemas operacionais fracamente acoplados para multicomputadores heterogêneos (LAN and WAN)	Oferecer serviços locais para clientes remotos
Middleware	Nível adicional em cima do SOR implementando serviços de propósito geral	Prover distribuição transparência

Revisão entre:

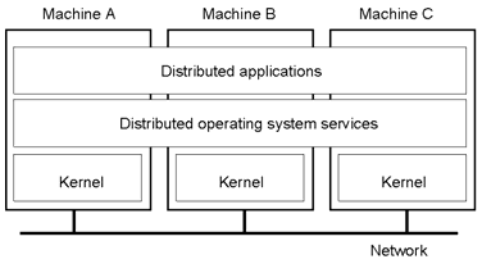
- SOD (DOS - Distributed Operating Systems)
- SOR (NOS - Network Operating Systems)
- Middleware

SO Uniprocessador



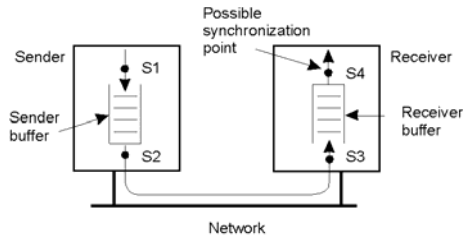
- Separação de aplicações do código de SO através de um microkernel.

SO Multicomputador (1)



Estrutura geral de um SO para um multicomputador

SO Multicomputador (2)



Alternativas para bloquear e buferizar usando passagem de mensagem.

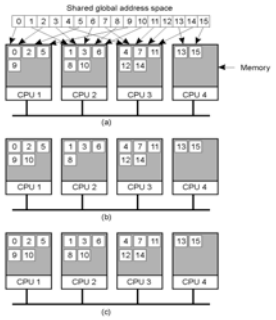
SO Multicomputador (3)

Pontos de Sincronização	Buffer enviante	com. confiável guarantida?
Bloquear enviante até buffer livre	sim	não necessariamente
Bloquear enviante até mensagem enviada	não	não necessariamente
Bloquear enviante até mensagem recebida	não	Necessario
Bloquear enviante até mensagem entregue	não	Necessario

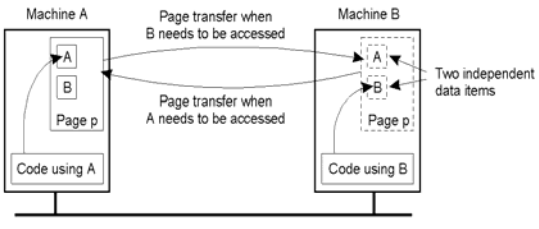
Relação entre bloqueio, bufferização e comunicação confiável.

Memória Distribuída Compartilhada (1)

- Páginas do espaço de endereçamento distribuídas entre 4 máquinas
- Situação depois CPU 1 referecia página 10
- Situação se página 10 é apenas leitura e replicação é usada

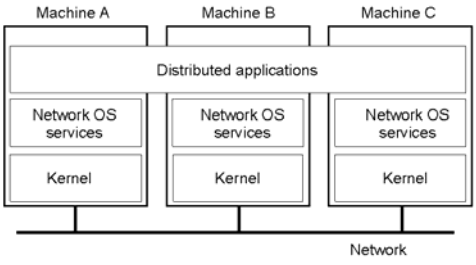


Memória Distribuída Compartilhada (2)



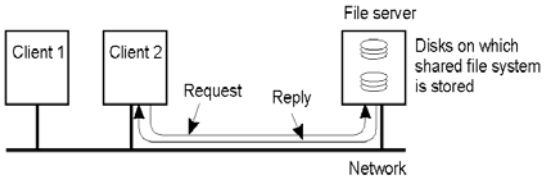
Falso compartilhamento de página entre 2 processos independentes.

Sistema Operacional de Rede (1)



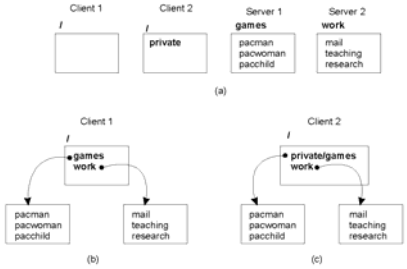
Estrutura geral de SOR.

Sistema Operacional de Rede (2)



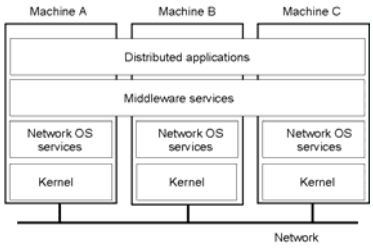
Dois clientes e um servidor em um SOR.

Sistema Operacional de Rede (3)



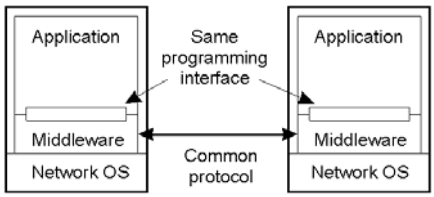
Diferentes clientes podem montar os servidores em diferentes lugares.

Middleware



Estrutura geral de um sistema distribuído com middleware.

Middleware



- Em um sistema distribuído aberto baseado em middleware, os protocolos usados por cada middleware deveriam ser os mesmos, assim como as interfaces que eles oferecem para as aplicações.

Middleware

- **Serviços de Comunicação**: abandona primitiva de passagem de mensagens baseada em sockets em favor de:
  - chamadas de procedimentos através da rede
  - invocação de métodos remotos
  - sistemas de fila de mensagens
  - comunicação com streams
  - serviço de notificação de eventos
- **Serviços de sistemas de informação**: serviços que ajudam gerenciar dados em SDs:
  - serviços de nomes
  - serviços de diretório (máquinas de busca)
  - serviços de localização
  - cache e replicação

Middleware

- **Serviços de Controle**: serviços para dar a aplicação controle sobre quando, onde e como acessar dados:
  - processamento de transações distribuídas
  - migração de código
- **Serviços de segurança**: serviços para comunicação e processamento seguro:
  - serviços de autenticação e autorização
  - serviços de criptografia
  - serviços de auditoria

Comparação entre Sistemas

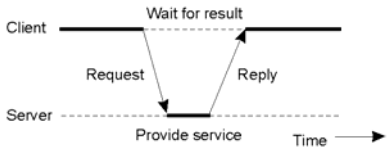
Item	SO Distribuído		SO rede	SO baseado em Middleware
	Multiproc.	Multicomp.		
Grau de transparência	Muita alta	Alta	Baixa	Alta
Mesmo SO em todos nodos	Sim	Sim	Não	Não
Numero de cópias de SO	1	N	N	N
Base para comunicação	Memoria comp.	Mensagens	arquivos	Específico do Modelo
Gerencia recursos	Global, central	Global, distribuída	Por nodo	Por nodo
Escalabilidade	Não	Moderada	Sim	Variável
Abertura	Fechado	Fechado	Aberta	aberta

Uma comparação entre SO multiprocessador, SO multicomputer, SO rede e SO baseado em middleware.

Modelo Cliente-Servidor

- Questão central na organização de SDs diz respeito a como organizar os *processos* no sistema. Existe uma certa concordância em pensar em termos de *clientes* que requisitam serviços de servidores.
  - Modelo básico
  - Níveis
  - Arquiteturas

Clientes e Servidores



Interação geral entre clientes e servidores.

Um exemplo Cliente / Servidor (1)

```
/* Definitions needed by clients and servers.                    */
#define TRUE             1
#define MAX_PATH       255   /* maximum length of file name      */
#define BUF_SIZE      1024   /* how much data to transfer at once */
#define FILE_SERVER    243   /* file server's network address     */

/* Definitions of the allowed operations */
#define CREATE           1   /* create a new file                */
#define READ             2   /* read data from a file and return it */
#define WRITE            3   /* write data to a file             */
#define DELETE           4   /* delete an existing file          */

/* Error codes. */
#define OK               0   /* operation performed correctly    */
#define E_BAD_OPCODE    -1   /* unknown operation requested      */
#define E_BAD_PARAM     -2   /* error in a parameter             */
#define E_IO            -3   /* disk error or other I/O error    */

/* Definition of the message format. */
struct message {
     long source;            /* sender's identity                */
     long dest;              /* receiver's identity              */
     long opcode;            /* requested operation              */
     long count;             /* number of bytes to transfer      */
     long offset;            /* position in file to start I/O    */
     long result;            /* result of the operation          */
     char name[MAX_PATH];    /* name of file being operated on   */
     char data[BUF_SIZE];    /* data to be read or written       */
};
```

header.h

Um exemplo Cliente / Servidor (2)

```
#include <header.h>
void main(void) {
     struct message ml, m2;          /* incoming and outgoing messages */
     int r;                          /* result code                    */

     while(TRUE) {                   /* server runs forever            */
          receive(FILE_SERVER, &ml);  /* block waiting for a message    */
          switch(ml.opcode) {         /* dispatch on type of request    */
               case CREATE:   r = do_create(&ml, &m2); break;
               case READ:     r = do_read(&ml, &m2); break;
               case WRITE:    r = do_write(&ml, &m2); break;
               case DELETE:   r = do_delete(&ml, &m2); break;
               default:       r = E_BAD_OPCODE;
          }
          m2.result = r;             /* return result to client        */
          send(ml.source, &m2);      /* send reply                     */
     }
}
```

- Servidor, geralmente fornece serviços relacionados a um recurso compartilhado.

Um exemplo Cliente / Servidor (3)

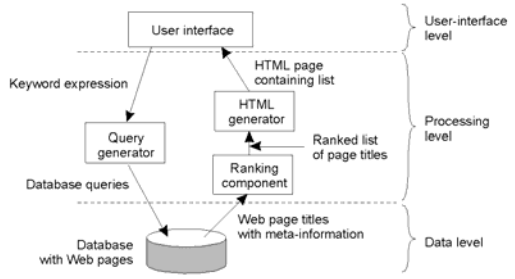
```
#include <header.h>
int copy(char *src, char *dst){          /* procedure to copy file using the server */
     struct message ml;                  /* message buffer                          */
     long position;                      /* current file position                   */
     long client = 110;                  /* client's address                        */

     initialize( );                      /* prepare for execution                   */
     position = 0;
     do {
          ml.opcode = READ;              /* operation is a read                     */
          ml.offset = position;          /* current position in the file            */
          ml.count  = BUF_SIZE;                                    /* how many bytes to read*/
          strcpy(&ml.name, src);         /* copy name of file to be read to message */
          send(FILESERVER, &ml);         /* send the message to the file server     */
          receive(client, &ml);          /* block waiting for the reply             */

          /* Write the data just received to the destination file.                  */
          ml.opcode = WRITE;             /* operation is a write                    */
          ml.offset = position;          /* current position in the file            */
          ml.count  = ml.result;         /* how many bytes to write                 */
          strcpy(&ml.name, dst);         /* copy name of file to be written to buf  */
          send(FILE_SERVER, &ml);        /* send the message to the file server     */
          receive(client, &ml);          /* block waiting for the reply             */
          position += ml.result;         /* ml.result is number of bytes written    */
     } while( ml.result > 0 );           /* iterate until done                      */
     return(ml.result >= 0 ? OK : ml.result);  /* return OK or error code          */
}
```

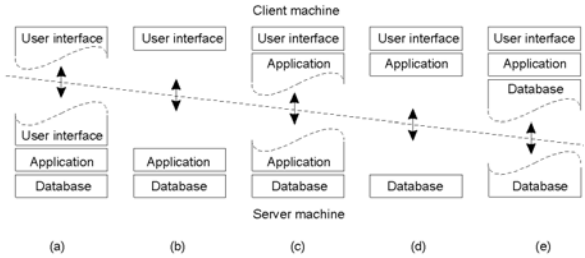
- Cliente, permite acesso a serviço remoto (API para serviços)

Nível de Processamento



- A organização geral de uma máquina de busca na Internet em 3 níveis diferentes

Arquiteturas Multitiered (1)



Organizations alternativas cliente-servidor (a) – (e).

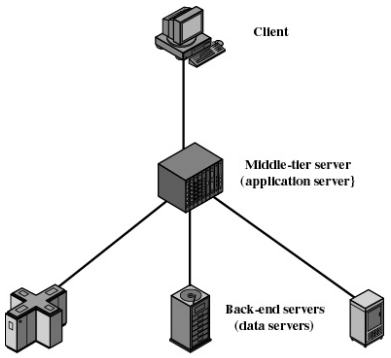
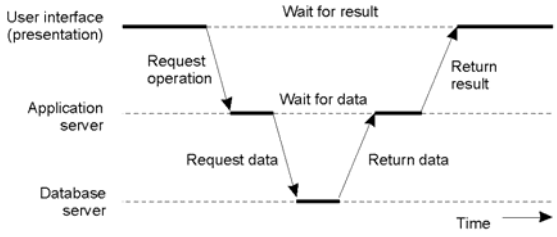


Figure 13.6 Three-tier Client/Server Architecture

Arquiteturas Multitiered (2)

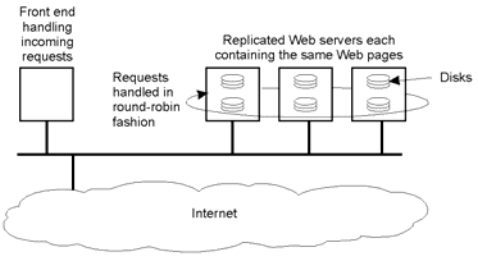


exemplo de um servidor agindo como um cliente.

Arquiteturas Alternativas

- **Servidores cooperantes**: serviço esta fisicamente distribuído em uma coleção de servidores:
  - arquiteturas multi-tiered tradicionais
  - sistemas de arquivos replicados
  - serviço de news
  - serviços de nomes (DNS, X.500)
  - serviços financeiros
- **Clientes cooperantes**: aplicação distribuída existe em virtude da colaboração de clientes:
  - teleconferência onde cada cliente tem uma estação
  - arquiteturas publica/subscreve nas quais o papel do cliente e servidor é confundido

Arquiteturas Modernas



exemplo de distribuição horizontal de um serviço Web.