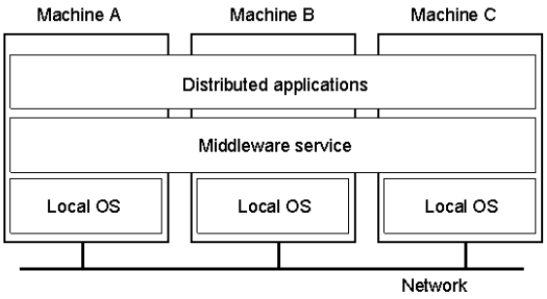
1) INTRODUÇÃO

Um sistema distribuído é: Uma coleção de computadores independentes que aparecem para o usuário como um único sistema coerente. Conforme mostra a Figura 1. A Figura 2 mostra os tipos de Transparências que um Sistemas Distribuídos pode ter.



Um sistema distribuído organizado como middleware.

Figura 1

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

Figura 2

Um SD também deve ter Transparência de Paralelismo, ou seja, não importa como o Sistema se expressa, a idéia essencial é que os usuários não devem ter que estar ciente da existência de múltiplas CPUs no sistema.

Exemplos de sistemas distribuídos: aplicações comerciais, aplicações de Internet, aplicações de vídeo-conferencia, Groupware.

2) MIDDLEWARE

Para suportar computadores e redes heterogêneas e, simultaneamente, oferecer uma visão de sistema único, os SDs costumam ser organizados por meio de uma camada de software – que é situada logicamente entre uma camada de nível mais alto, composta de usuários e aplicações, e uma camada subjacente, que consiste em sistemas operacionais e facilidades básicas de comunicação, conforme mostra a Figura 3. Tal sistema distribuído é denominado Middleware.

A camada de Middleware se estende por varias maquinas e oferece a mesma interface a cada aplicação.

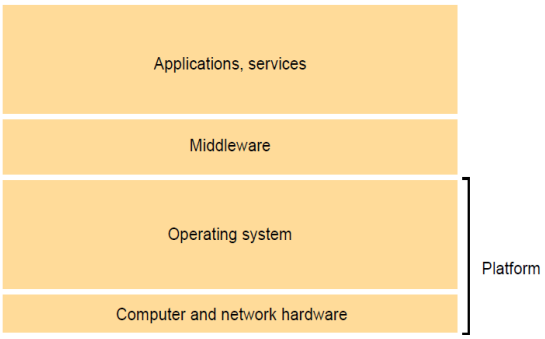
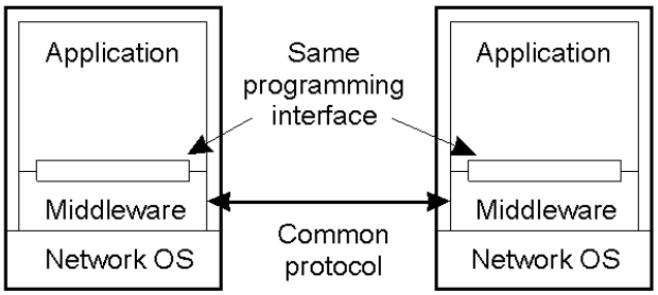


Figura 3

Sistema Operacional esconde o hardware, o Middleware também, mas em um nível mais alto. O middleware forma uma camada entre aplicações e plataformas distribuídas com a finalidade de proporcionar um grau de transparência de distribuição de dados, processamento e controle.

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. Conforme mostra a Figura abaixo.



Serviços fornecidos por um Middleware:

A) Serviço de comunicação → Socket, RPC, RMI, Streams e Notificações de Eventos.
B) Serviços de sistemas de informação → DNS, Máquina de busca, Localização e cache
C) Serviços de Controle → serviços para dar a aplicação controle sobre quando, onde e como acessar dados: Para processamento de transações distribuídas e migração de código
D) Serviços de segurança → serviços para comunicação e processamento seguro: Autenticação e autorização, Criptografia, Auditoria e Assinatura Digital

3) ARQUITETURAS

A Figura 4 mostra clientes invocando servidores individuais. Um servidor pode ser cliente de outro servidor. A Figura 5 um serviço provido por múltiplos servidores.

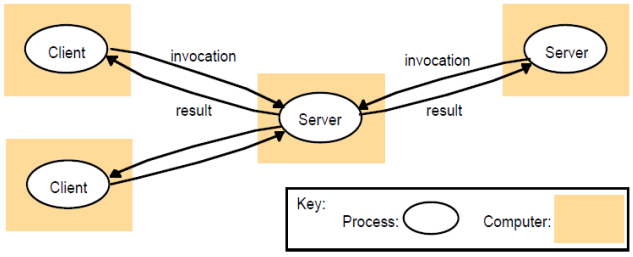


Figura 4 – Cliente/Servidor

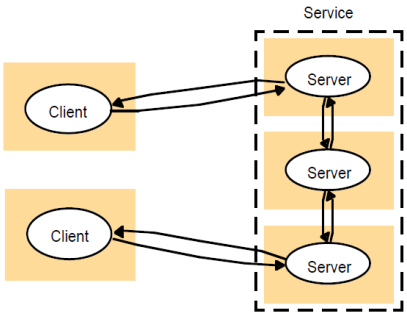


Figura 5

As vantagens da arquitetura da Figura 5 são: O SO é ativado de acordo com a necessidade do cliente, a transparência da migração é garantida. Cada servidor tem uma intancia do SO.

A Figura 6 mostra clientes leves e servidores de computação. Exemplo desse tipo de arquitetura são so computadores com acesso a internet do 4º e 6º andar da UERJ. O termo cliente "leve" se refere a uma camada de software, em um computador local, que oferece ao usuário uma interface baseada em janelas para que este possa executar programas aplicativos em um computador remoto.

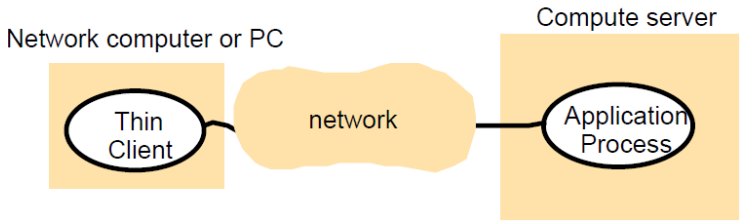


Figura 6

A Figura 7 mostra servidores assistidos por um proxy. Um servidor proxy é um computador intermediário que fica entre o computador do usuário e a Internet. Pode ser utilizado para registrar o uso da Internet e também para bloquear o acesso a um site da Web. O firewall do servidor proxy bloqueia alguns sites ou páginas da Web por vários motivos.

A Figura 8 mostra uma aplicação Peer-to-peer (par a par) que é um tipo de rede de computadores onde cada estação possui capacidades e responsabilidades equivalentes, ou seja, em um momento é cliente e em outro é servidor. Isto difere da arquitetura cliente/servidor, no qual alguns computadores são dedicados a servirem dados a outros.

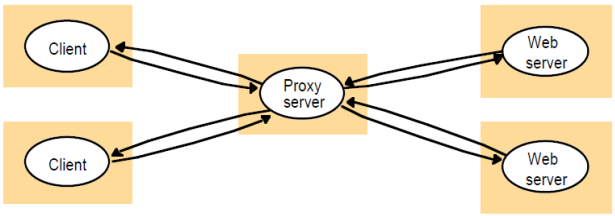


Figura 7 – Servidor Proxy-Web

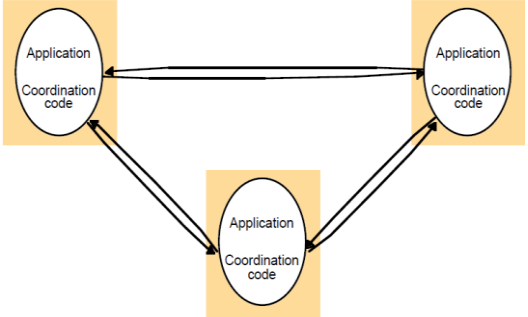


Figura 8 – Peer-to-peer

4) APPLET NA WEB

Os applets representam um exemplo bem conhecido e bastante utilizado de código móvel – o usuário, executando um navegador, seleciona um link que aponta para um applet, cujo código é

armazenado em um servidor web. Conforme mostra a Figura 9 . O código é carregado no navegador e, como se ve na Figura 10, posteriormente executado.

Cliente interage com o servidor Web e realiza o download do código do applet.

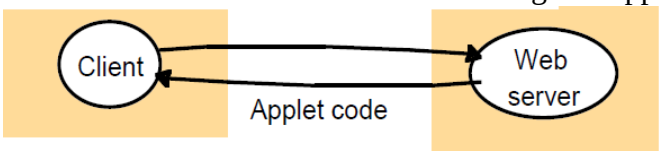


Figura 9

Cliente interage com o applet localmente.

Figura 10

Uma vantagem do applet é que ele pode dar uma boa resposta interativa, pois não sofre atrasos, nem a variação da largura de banda associada a comunicação na rede.

5) REDES ESPONTÂNEAS

As redes espontâneas, também conhecidas por redes sem infra-estrutura ou redes ad-hoc, não possuem nenhum tipo de estação-base ou roteador. Todos os nós são móveis e podem conectar-se, dinamicamente, uns com os outros, formando a rede espontaneamente. Cada nó da rede funciona como um roteador que descobre e mantém rotas para outros nós. Aplicações típicas para essas redes incluem operações de resgate em desastres e encontros ou convenções em que pessoas precisam compartilhar informações rapidamente. A Figura 11 mostra uma Rede Espontanea.

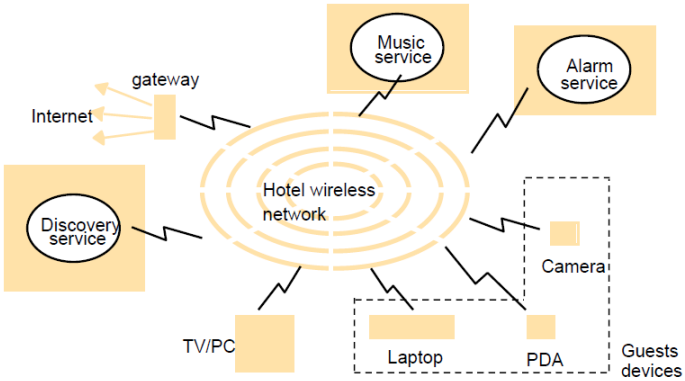


Figura 11

Ao se conectar a uma rede espontânea o computador deve ser capaz de acessar primeiro o Discovery Service dessa rede.

6) ESCALABILIDADE

Escalabilidade é uma característica desejável em todo o sistema, em uma rede ou em um processo, que indica sua habilidade de manipular uma porção crescente de trabalho de forma uniforme, ou estar preparado para crescer.

Ela pode ser medida segundo, no mínimo, três dimensões diferentes: Tamanho (ser fácil adicionar usuários e recursos ao sistema), Termos Geográficos (usuários e recursos podem estar longe uns dos outros) e Termos Administrativos (ser fácil de gerenciar, mesmo que abranja muitas organizações diferentes).

A Figura a seguir mostra os problemas de escalabilidade de um SD.

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

Figura 12

Solução para os problemas de escalabilidade

1) Evitar componentes centralizados, dados centralizados, algoritmos centralizados, etc.
2) Dar preferência a algoritmos não centralizados, ou seja, nenhuma máquina tem a informação completa sobre o estado do sistema; Cada máquina toma decisões com base apenas em sua informação local; Falha em uma máquina não arruinará o algoritmo e Não assumir que existe um "clock global"

Técnicas de escalabilidade

1) **Distribuição** : dividir dados e computações nas máquinas.
Exemplo: DNS que é uma hierarquia em árvore de domínios, dividida em zonas se sobreposição.

2) **Replicação** : fazer cópias disponíveis em várias máquinas.
3) **Caching** : permitir processos clientes acessar cópias locais.

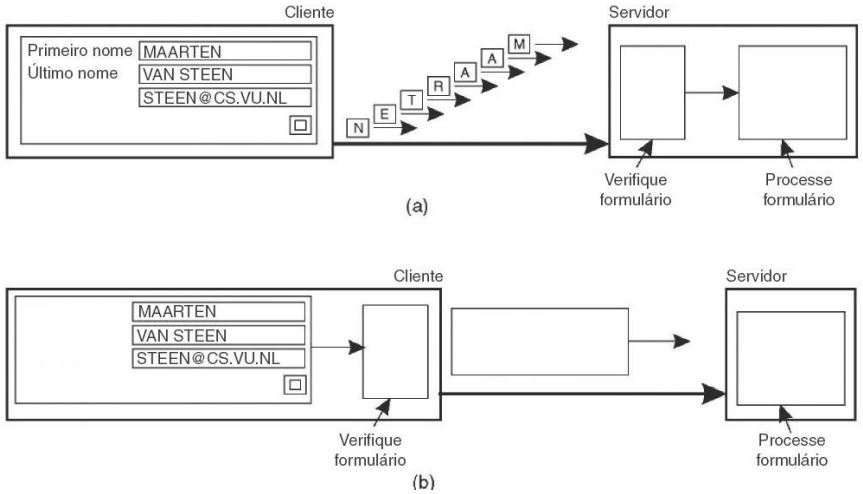


Figura 1.2 A diferença entre deixar (a) um servidor ou (b) um cliente verificar formulários à medida que são preenchidos.

7) CILADAS

Premissas falsas adotadas ao se desenvolver pela primeira vez uma aplicação distribuída:

– Rede é confiável, segura e homogênea
– Topologia constante
– Latência e Custo de Transporte zero
– Largura de banda é infinita
– Existe somente um administrador

8) TIPOS DE SISTEMAS DISTRIBUIDOS

A) Sistemas de Computacão - Computação de Cluster

Hardware consiste em um conjunto de estações de trabalho ou Pcs semelhantes a conexão é feita através de uma rede local. Em quase todos os casos, a computação de cluster é usada para programação paralela na qual um único programa é executado em paralelo.

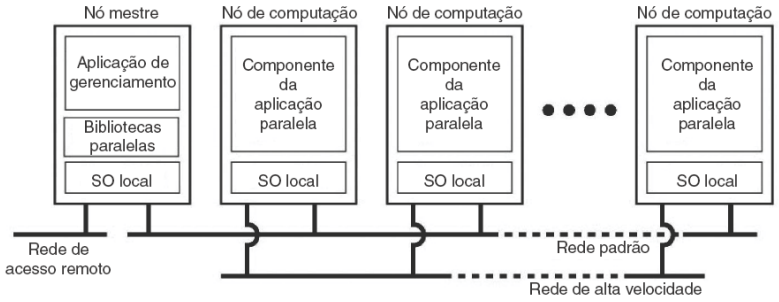
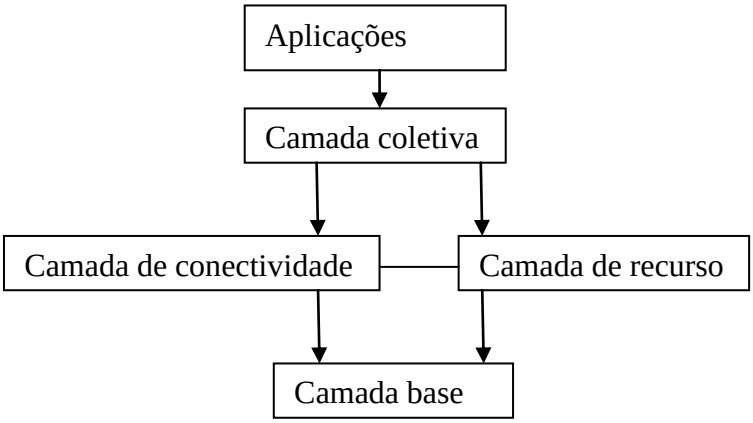


Figura 1.4 Exemplo de um sistema de computação de cluster.

B) Sistemas de Computacão - Computação em Grade

Tem Heterogeneidade. Recursos de diferentes organizações são reunidos para permitir a colaboração de um grupo de pessoas ou instituições.



C) Sistemas de Informação

Sistemas empresariais desenvolvidos para integrar diversas aplicações individuais, onde a interoperabilidade se mostrou "dolorosa". Exemplos: Sistemas de processamento de Transações e Integração de Aplicações Empresariais.

D) **Sistemas Pervasivos**

Instabilidade é o comportamento esperado destes sistemas. Dispositivos de computação móveis e embutidos (Pequenos, Alimentação por bateria, Mobilidade e Conexão sem fio). Ausência geral de controle administrativo humano.

Requisitos para as aplicações pervarsivas são: Adotar mudanças contextuais, Incentivar composição ad hoc e Reconhecer compartilhamento como padrão.

Tipos de Sistemas Pervasivos: Sistemas Domésticos, Sistemas Eletrônicos para Tratamento de Saúde e Redes de Sensores.

9) Conceitos de Hardware

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

10) 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

11) SO MULTICOMPUTADOR

A Figura 13 mostra o buffer de envio e de recebimento de uma requisição utilizando troca de mensagem. A Figura 14 mostra os possíveis pontos de sincronização.

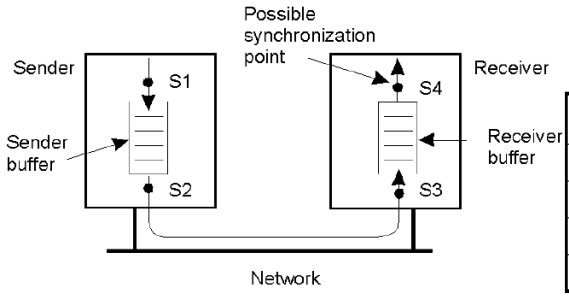


Figura 13

Pontos de Sincronização	Buffer enviante	com. confiável garantida?
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

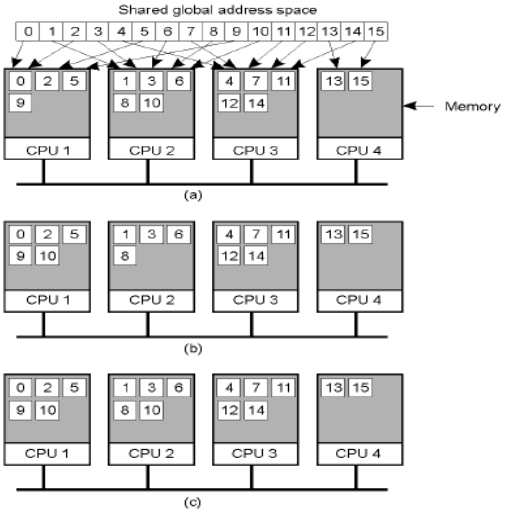
Figura 14

12) MEMÓRIA DISTRIBUÍDA COMPARTILHADA

a) Páginas do espaço de endereçamento distribuídas entre 4 máquinas

b) Situação depois CPU 1 referecia página 10

c) Situação se página 10 é apenas leitura e replicação é usada



Manter a coerência da memória distribuído e compartilhada não é trivial.

Por que usar memoria compartilhada? Para que dois processos em maquinas distintas consigam se comunicar como se estivessem na mesma maquina.

Qual o maior problema na utilização da memoria compartilhada? Sincronização. Garantir a segurança dos dados.
A Figura 15 mostra um falso compartilhamento de paginas entre dois processos independentes. Quando o processo B precisa de uma das paginas do processo A ela é enviada para ele e vice-versa.

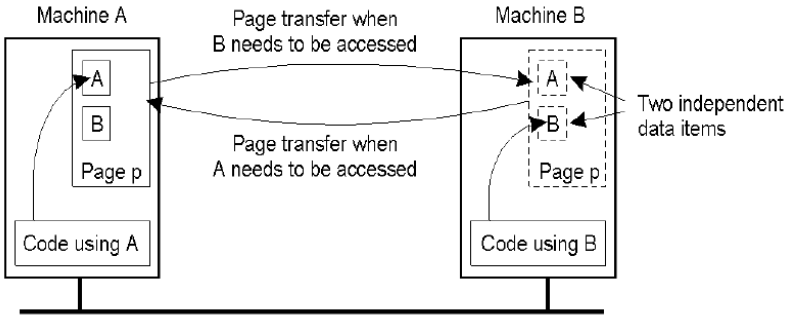


Figura 15

13) SISTEMA OPERACIONAL DE REDE

A Figura 16 mostra um SOR.

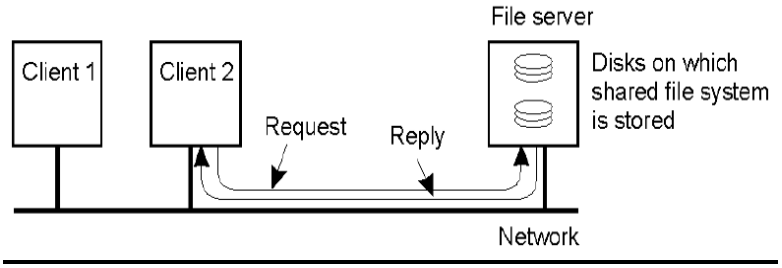


Figura 16

14) MODELO CLIENTE-SERVIDOR

Organização dos processos nos sistemas distribuídos. A Figura 17 mostra a representação da interação geral de clientes e servidores. O cliente faz um requisição e espera pelo resultado. O servidor recebe esta requisição, processa e envia o resultado. O cliente recebe o resultado e continua sua execução.

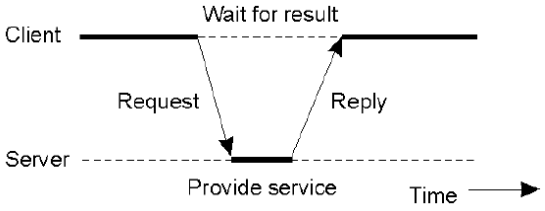


Figura 17

Um **servidor** é um processo que implementa um serviço especifico, por exemplo, um serviço de bando de dados. O Servidor, geralmente fornece serviços relacionados a um recurso compartilhado.

Um **cliente** é um processo que requisita um serviço de um servidor e espera pela resposta.

A Figura 18 apresenta arquiteturas alternativas cliente-servidor. Arquiteturas Multitiered.

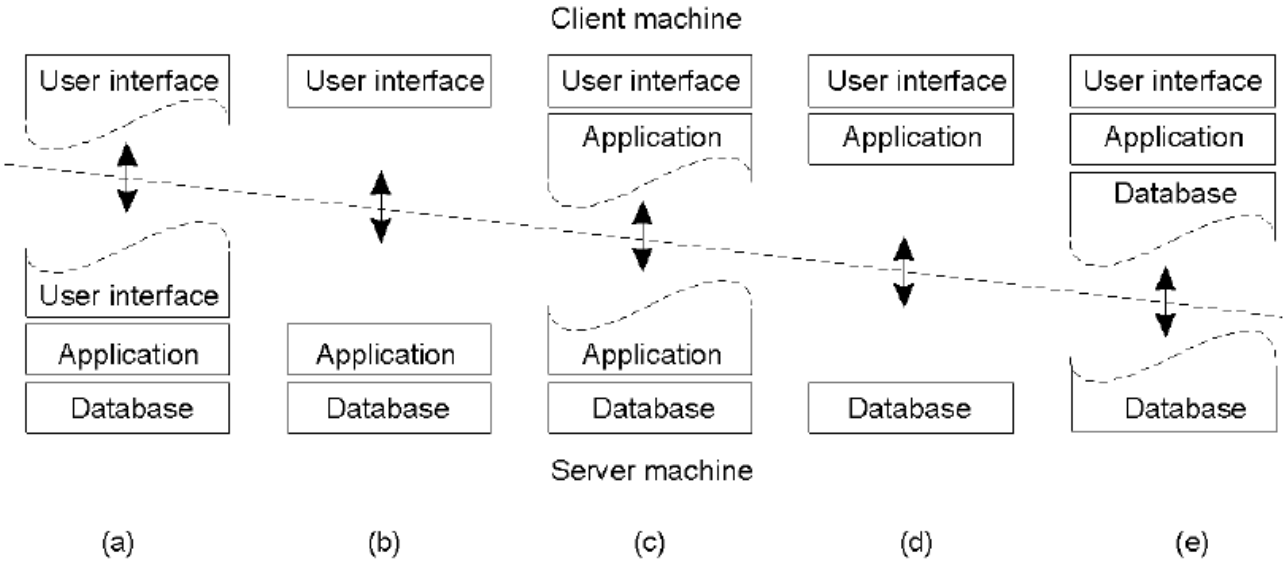


Figura 18

Exemplos de sistemas que utilizam as arquiteturas da figura acima:

(a)
(b) Web simples (site)
(c)
(d)
(e) Imposto de Renda

A organização geral de uma máquina de busca na Internet contem 3 níveis diferentes: Interface com o usuário, Processamento e Banco de dados.

Servidores cooperantes ocorrem quando o serviço esta fisicamente distribuído em uma coleção de servidores, por exemplo, arquiteturas multitiered e serviços de nomes (DNS)

Clientes cooperantes ocorrem quando a aplicação distribuída existe em virtude da colaboração de clientes, por exemplo, teleconferência onde cada cliente tem uma estação.

15) COMUNICAÇÃO EM REDE

A camada de Transporte oferece um serviço de entrega de um processo origem a um processo de destino. Situada entre a camada de Rede e a camada do Middleware sua finalidade é um transporte fim-a-fim para as aplicações de forma independente da internet de suporte. Possui Interface para o uso da rede por aplicações que precisam se comunicar com aplicações remotas. Só necessitam saber o endereço da máquina e o tipo de serviço destinatário (IP + socket).

Existem dois tipos de protocolos de transportes utilizados na prática: TCP/IP e UDP.

TCP/IP (Transmission Control Protocol/ Internet Protocol) → Serviço de entrega confiável, garante ordenação, estilo Circuito Virtual (Estabelece o circuito, troca de mensagens e encerra o circuito), multiplexação, detecção e correção de erros e interface de programação por sockets.

UDP (User Datagram Protocol) → Características de serviço de entrega não confiável, não garante ordenação, só acrescenta multiplexação e detecção de erro ao IP, interface de programação por sockets

16) SOCKETS

Definem pontos terminais de acesso à comunicação. Cada socket é definido por um endereço IP concatenado a um número identificador de porta. Clientes e servidores conectam-se para comunicação através de portas.

Um socket por ter: interações simples (rápido, datagrama (UDP)) ou Interações longas (garantia de entrega, circuito virtual (TCP)).

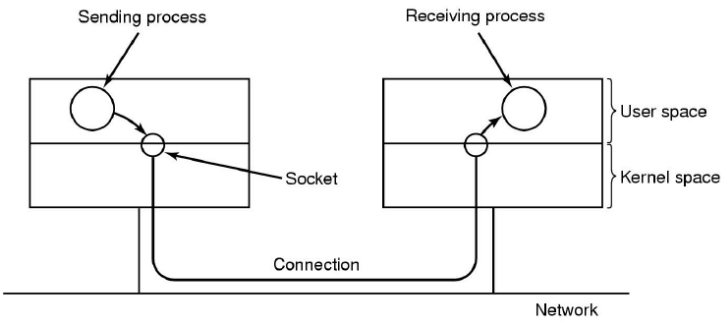


Figura 19

Desvantagens do RPC e RMI:
– Componentes da comunicação devem estar ligados e em funcionamento
– Precisam saber exatamente como se referir um ao outro

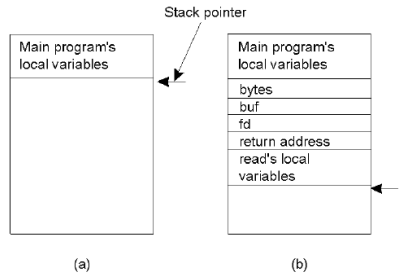
17) RPC (Remote Procedure Call - Chamada a Procedimentos Remotos)

Programar utilizando primitivas de comunicação não é transparente a ótica de muitos projetistas é a de tornar Sistemas Distribuídos parecidos com Sistemas Centralizados. Primitivas orientadas a E/S não conseguem produzir este efeito.

Idéia simples é permitir que um processo chame um procedimento que esteja localizado em outra máquina. O objetivo é não utilizar nenhuma primitiva de E/S (send/receive) pelo programador. Ao contrario do socket que o programador precisa utilizar send/receive.

O suporte a operação do RPC, em realidade, trabalha com troca de mensagens. A execução do procedimento é feita em outra máquina com espaço de endereçamento diferente. As máquinas podem ter representações diferentes para os tipos básicos (inteiros, float, etc.)

O RPC gera protocolos que identificam o tipo de maquina em que se esta trabalhando. A diferença entre duas aplicações que chama o servidor está nos procedimentos e na interface que são diferentes, mas o mecanismo é o mesmo. O Rpcgen gera os arquivos que compõem o Middleware do RPC.



a) Passagem de parâmetro chamada local: pilha antes da chamada
b) Pilha enquanto procedimento está ativo

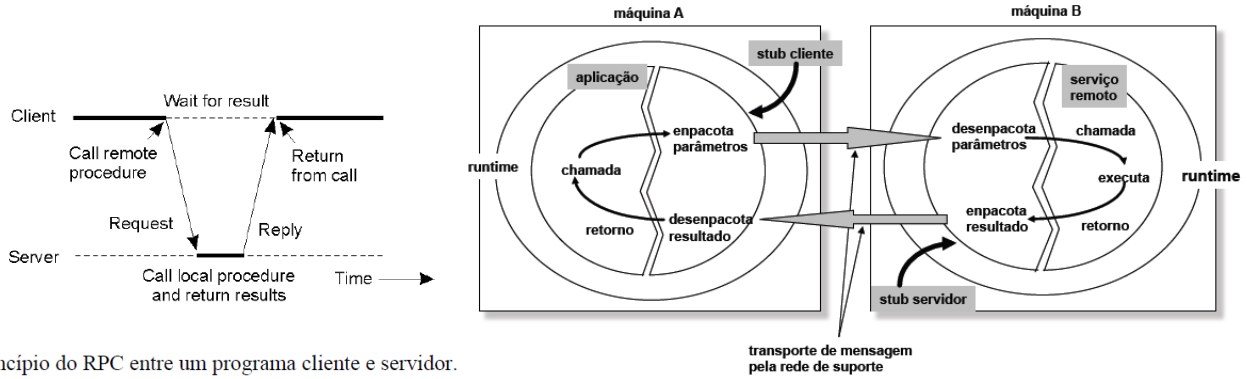
Para entender como uma RPC funciona, em primeiro lugar é importante entender completamente como funciona uma chamada de procedimento convencional, isto é, em uma única maquina. Considere um chamada em C como count = read(fd, buf, nbytes); onde fd é um inteiro que indica um arquivo, buf é um vetor de caracteres no qual os dados são lidos, e nbytes é um outro inteiro que informar quantos bytes ler.

Se a chamada for feita pelo programa principal, a pilha vai estar como mostra a Figura acima em (a) antes da chamada. Para fazer a chamada, o chamador passa os parâmetros para a pilha em ordem, começando pelo ultimo, como mostra a Figura em (b). Após a conclusão da execução do procedimento read, ele coloca o valor de retorno em um registrador, remove o endereço de retorno e devolve o controle ao chamador. A diferença entre chamadas por valor e por referencia é muito importante para RPC.

A Chamada de procedimento remoto e invocação remota de objetos contribuem para esconder a comunicação em sistemas distribuídos, geralmente baseado no modelo C-S (síncrono). Algumas vezes nenhum destes mecanismos é adequado.

Não podemos assumir que o lado recebedor está executando quando a requisição é emitida. A natureza síncrona de RPC e RMI não é adequada, cliente não pode fazer outra coisa enquanto espera. Algumas vezes o mecanismo adequado é mensagens, ou seja, o modelo síncrono não é apropriado (mail, news).

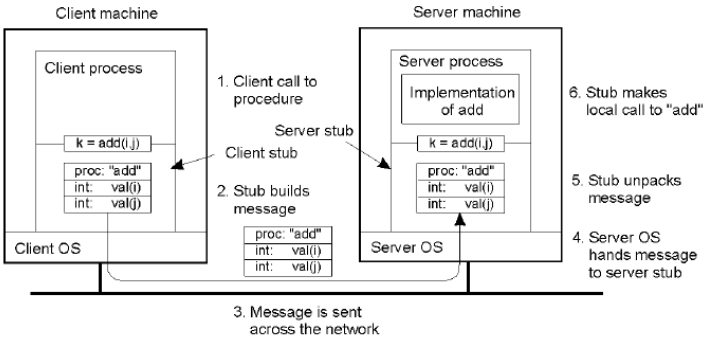
Um apêndice (stub) é um pedaço de código que transforma requisições que vêm pela rede em chamadas de procedimento locais. O cliente tem o apêndice de cliente e o servidor tem o apêndice de servidor.



Princípio do RPC entre um programa cliente e servidor.

Passos do RPC da Figura lado direito Acima

1. Cliente chama stub cliente.
2. Stub client constrói mensagem, chama SO local:
    2.1. SO cliente envia mensagem para SO remoto
    2.2. SO remoto entrega mensagem para stub servidor
3. Stub servidor retira parametros, chama servidor.
4. Servidor realiza trabalho, retorna resultado para stub.
5. Stub servidor coloca na mensagem, chama SO local.
    5.1. SO servidor envia mensagem para SO cliente
    5.2. SO cliente entrega mensagem para stub cliente
6. Stub retira resultado, retorna para cliente



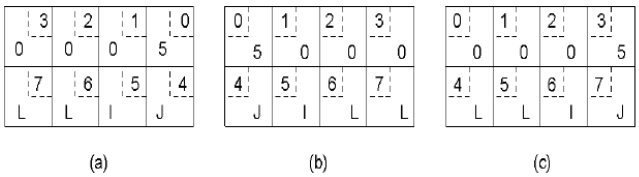
Trocando parâmetros em RPC

A função do apêndice de cliente é pegar seus paramentros, empacota-los em uma mensagem e envia-los ao apêndice de servidor.

A Figura ao lado mostra a Passagem de parâmetros de valor através do RPC.

Contanto que as maquinas cliente e servidor sejam idênticas e todos os parâmetros e resultados sejam tipos escalares esse modelo funciona bem.

Buffer que armazena os parâmetros. Números indicam endereço



a) Mensagem original no Pentium
b) Mensagem depois de recebida em SPARC
c) Mensagem depois de invertida

Em sistemas distribuídos de grande porte, é comum estarem presentes vários tipos de maquinas. Cada maquina constuma ter sua própria representação para numero e outros itens de dados.

Algumas maquinas, como Intel numeram seus bytes da direita para a esquerda (denominado **little endian**), enquanto outras, como a SPARC, os numeram ao contrario (denominado **big endian**). Exemplo na Figura ao lado.

18) IDL (Interface Definition Language)

Uma interface consiste em um conjunto de procedimentos que podem ser chamados por um cliente e que são implementados por um servidor. Interfaces constumam ser especificadas por meio de uma linguagem de programação de interface (IDL).

Exemplo de RPC IDL

```
/* MSG.X: Especificacao do servidor "printmessage" para RPC */
#define VERSION_NUMBER 1
program MESSAGE_PROG {
        version PRINT_MESSAGE_VERSION {
                int PRINTMESSAGE(string) = 1;
        } = VERSION_NUMBER;
} = 0x20000001;
```

Exemplo de RPC (Server)

```
#include <stdio.h>
#include "msg.h" //generated by rpcgen
int * printmessage_1_svc(char ** msg, struct svc_req *req)
{
        static int result;
        result = fprintf(stdout," %s\n", *msg);
        return (&result);
}
```

Exemplo 1 de RPC (Client)

```
#include "msg.h"
int main(int argc, char **argv) {
        CLIENT *clnt; int *result;
        char *server; char *message;
        server = argv[1]; message = argv[2];
        clnt = clnt_create(server, MESSAGE_PROG, PRINT_MESSAGE_VERSION,
"udp");
        result = printmessage_1(&message, clnt); //remote call
        printf("Message delivered to %s\n",server);
        clnt_destroy( clnt );
        return 0;
}
```

19) DYNAMIC BINDING

O cliente precisa localizar o servidor que vai executar o procedimento. Os tipos de dados precisam ser os mesmos no cliente e servidor. O cliente e o servidor são compilados separadamente, mas precisam ter a mesma interface.

RPC utiliza uma porta "alta" não utiliza uma porta vazia "sem uso". Depois que aloca uma porta para um processo fica nela até o processo terminar.

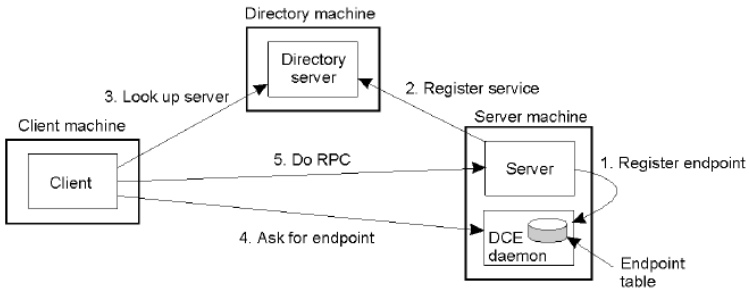
Binder → Tabela com nomes + endereços dos serviços correntemente exportados. O checksum que identifica a versão da interface exportada.

Instância de um serviço começa a ser executado, a interface é exportada. Um checksum é calculado a partir da interface exportada. O cliente ao iniciar, precisa se ligar (bind) ao serviço remoto desejado e envia uma mesagem "Onde Está?" ao Binder.

20) VINCULACAO DE UM CLIENTE A UM SERVIDOR

Para pemitir que um cliente chame um servidor, é necessário que o servidor seja registrado e esteja preparado para aceitar chamadas que chegam. O registro de um servidor possibilita que um cliente localize e se vincule a um servidor. A localização é feita em duas etapas: Localizar a maquina do servidor e Localizar o servidor (o processo correto na maquina do servidor).

Conforme mostra a Figura abaixo, na segunda etapa de localização do servidor o cliente precisa conhecer uma porta na maquina do servidor para enviar as mensagens. Uma porta (ou terminal) é usada pelo SO do servidor para distinguir mensagens que chegam de diferentes processos.



1 → Registrar porta
2 → Registrar serviço
3 → Consultar servidor
4 → Solicitar porta
5 → Fazer RPC

No Linux o DCE daemon é o PortMap.

Em DCE, uma tabela de pares (servidor, porta) é mantida em cada maquina servidores por um processo denominado daemon DCE. Antes de ficar disponível para requisições que chegam, o servidor deve solicitar uma porta ao SO. Então, ele registra essa porta ao daemon DCE que por sua vez registra essa informação (incluindo quais protocolos o servidor fala (TCP/IP ou UDP)) na tabela de portas para utilização futura.

O servidor também se registra no serviço de diretório fornecendo-lhe o endereço de rede da maquina servidora e um nome sob o qual o servidor possa ser procurado.

21) SITUAÇÕES DE FALHA DO RCP

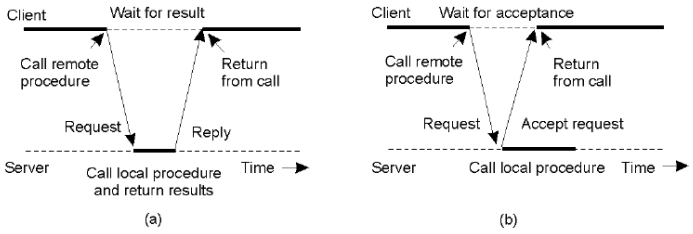
- Cliente não consegue localizar o servidor;
- Mensagens de Request perdidas: número de seqüência aos pedidos e descarta duplicatas;
- Servidor falha: maybe (nada garante), at least once (repete até conseguir ao menos uma vez) e at most once (tenta apenas uma vez e reporta ao cliente a ocorrência de falha), exactly once (cliente falha)

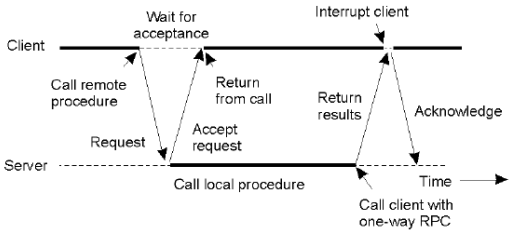
22) RCP ASSINCRONA E SINCRONA

RCP Sincrona é quando um cliente chama um procedimento remoto, o cliente bloqueia até que uma reposta seja retornada. Em RPC Assincrona o cliente continua imediatamente após emitir a resquisição RPC e o servidor enviar uma resposta de volta ao cliente informando que ela foi recebida e chama o procedimento requisitado.

RPC Assíncrona (1)

RPC Assíncrona (2)

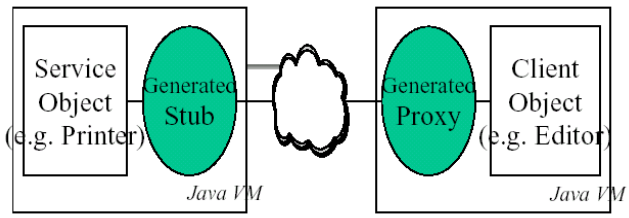




a) Interconexão entre cliente e servidor RPC tradicional
b) Interação usando RPC assíncrona cliente e servidor interagindo com RPC assíncrona

23) JAVA RMI (Remote Method Invocation)

Remote Method Invocation (RMI) é um mecanismo próprio de Java, similar a RPC. RMI permite um programa Java em uma máquina invocar um método remoto em um objeto remoto. Esse mecanismo Java, que gera as classes proxy de forma automática. O usuário codifica os objetos cliente e serviço.



O compilador RMI gera o código (stub e skeleton) responsável pela comunicação na rede. Invocação de métodos como fundamento. Obtém referência do objeto remoto. Invoca métodos diretamente ao objeto remoto.

Exemplo: Definição da Interface Remota

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Hello extends Remote {
        String sayHello() throws RemoteException;
}
```

Semelhante a IDL (Interface Definition Language) em RPC. Apenas define a interface do método remoto "sayHello()".

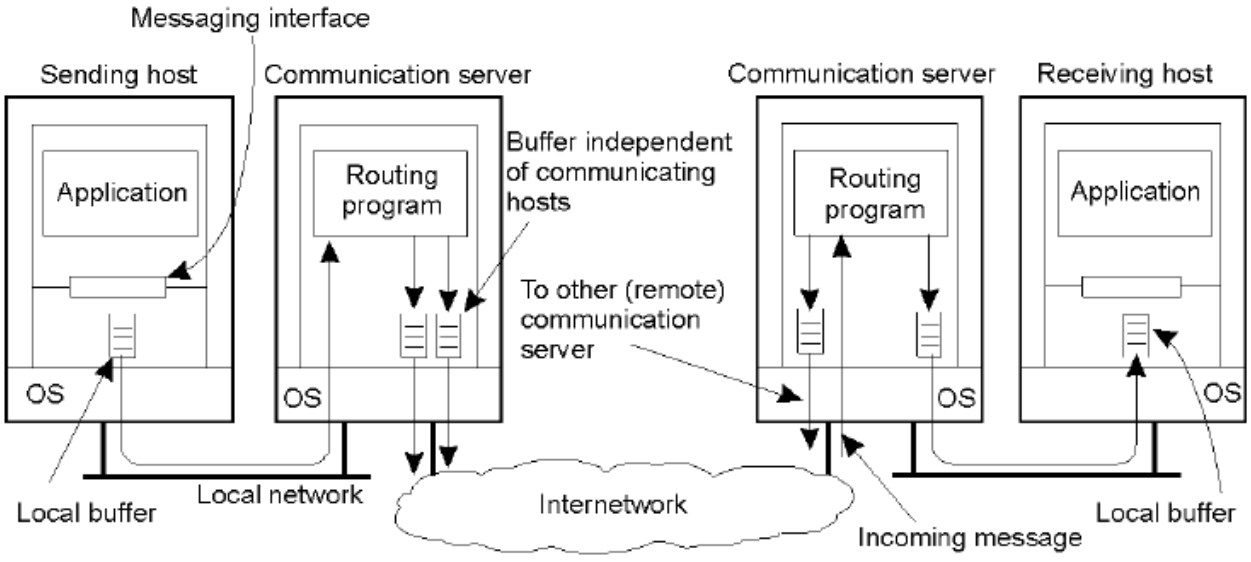
```
public class Server implements Hello
{
        public Server() {}
        public String sayHello(){
                return "Hello, distributed world!\n";
}
public static void main(String args[]) throws Exception{
        Server obj = new Server();
        Hello stub = (Hello) UnicastRemoteObject.exportObject(obj,
0);
        Registry registry = LocateRegistry.getRegistry();
        registry.bind("Hello",stub);
        System.out.println("Server ready"); } }
```

```
public class Client {
        private Client() {}
        public static void main(String args[]) throws Exception
        {
                String host = (args.length < 1) ? null : args[0];
                if(host == null){
                        System.err.println("Host is invalid!\n");
                        System.exit(1);
        }
                Registry registry = LocateRegistry.getRegistry(host);
                Hello stub = (Hello) registry.lookup("Hello");
                System.out.println("response: " + stub.sayHello());
        }
}
```

Compilando e executando
$ javac Client.java Server.java Hello.java
Na máquina servidora: $ rmiregistry & $ java Server
Na máquina cliente: $ java Client localhost

24) COMUNICACAO ORIENTADA A MENSAGENS

AS chamadas de procedimento remoto (RPC) e de invocações de objeto remoto (Java RMI) aprimoram a transparência de acesso. Nem sempre são adequados. Não se pode adotar como premissa de que o lado receptor está executando no momento em que uma requisição é emitida.



Organização de um sistema de comunicação genérico

A interface Sockets foi projetada para comunicação por redes que usam pilhas de protocolos de uso geral, tal como TCP/IP.

Sockets, RPC, RMI, etc. Não é considerado adequado para redes de interconexão de alto desempenho.

Características mais avançadas são desejadas: diferentes formas de buffer e sincronização.

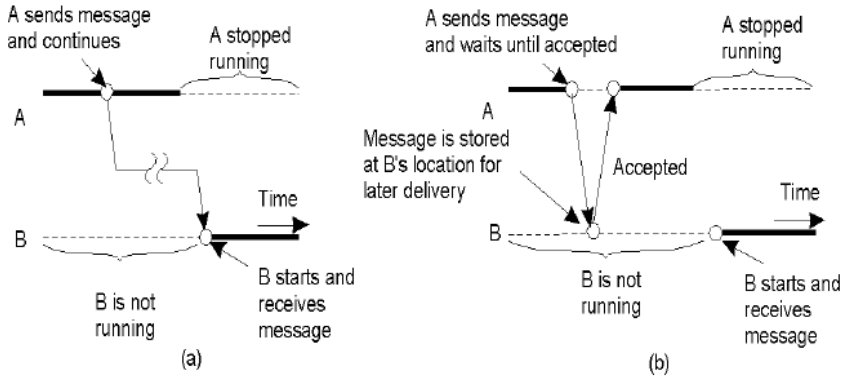
25) TIPOS DE COMUNICAÇÃO

A comunicação entre processos pode ser: **Síncrona** e **Assíncrona**. As primitivas de serviço podem ser do tipo: **Bloqueadas** e **Não bloqueadas**. E a comunicação segue dois modelos: **Persistente** e **Transiente**.

Comunicação síncrona → Os processos atendem a uma semântica de sincronização. O processo transmissor estabelece uma conexão e aguarda a resposta do processo remoto depois de enviada a solicitação. Ou seja, o enviante é bloqueado até sua msg ser armazenada em buffer local no recebedor ou realmente entregue. A forma mais forte é quando o enviante fica bloqueado até o recebedor processar a msg.

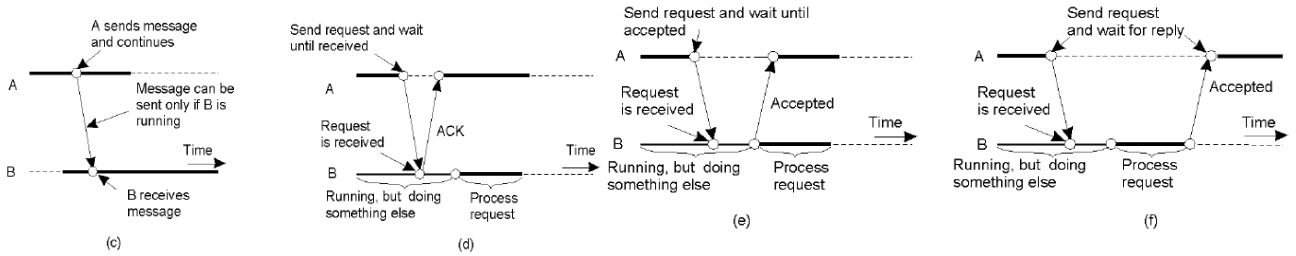
Comunicação assíncrona → O processo transmissor não aguarda a resposta do servidor. Depois de bufferizada a mensagem para posterior envio, o processo segue, sem aguardar a resposta do processo remoto. Ou seja, o enviante continua depois de submeter msg. A msg é armazenada em buffer local no enviante ou no servidor de comunicação.

Comunicação persistente → A mensagem submetida para transmissão é armazenada nos sistemas de comunicação e assim que possível é entregue ao seu destino, não sendo necessário que os processos "send" e "receiver" estejam rodando. Ou seja, mensagem submetida para transmissão é armazenada no sistema de comunicação até entregar para o recebedor. Aplicação enviante não precisa continuar executando e a aplicação recebedora não precisa estar executando na sub.



a) Comunicação persistente assíncrona
b) Comunicação persistente síncrona

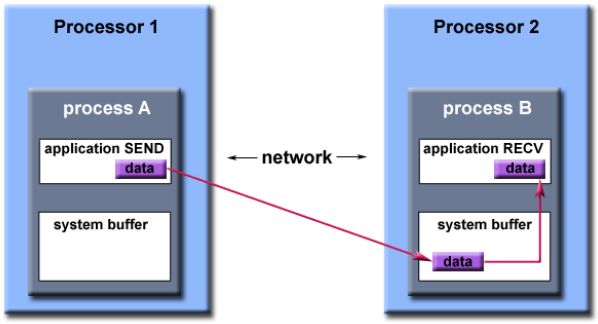
Comunicação transiente → A mensagem submetida para transmissão é armazenada nos sistemas de comunicação, se e somente se, os processos "send" e "receiver" estiverem rodando. Ou seja, mensagem é armazenada apenas enquanto as aplicações enviante e recebedora estão executando. Se não é possível entregar a msg ao próximo servidor a msg é descartada. O serviço de transporte, se o roteador não pode entregar a msg para próximo (destino), descarta msg.



c) Comunicação transiente assíncrona - UDP
d) Comunicação transiente síncrona, baseada na recepção
e) Comunicação transiente síncrona, baseada na entrega
f) Comunicação transiente síncrona, baseada na resposta - RPC

26) MPI (Interface de troca de mensagens)

Surgiu da consideração de que Sockets são insuficientes para escrever com facilidade aplicações de alta eficiência (HPC – High Performance Computing), pois apresentam problemas de abstração, suportando apenas primitivas simples "send" e "receive". E projetado para protocolos de pilha de protocolos de uso geral (TCP/IP), sendo considerados inadequados para protocolos proprietários para redes de alta velocidade.



A MPI foi projetada para aplicações paralelas. Ela faz uso direto da rede subjacente. Além disso, considera que falhas serias, como quedas de processos ou partições de rede, sejam fatais e não requeiram recuperação automática. A MPI adota a premissa de que a comunicação ocorre dentro de um grupo conhecido de processos.

Primitiva	Significado
MPI_bsend	Anexa mensagem de saída à um buffer local de envio
MPI_send	Envia uma mensagem e espera até copiar para buffer local ou remoto
MPI_ssend	Envia uma mensagem e espera até recebedor iniciar
MPI_sendrecv	Envia uma mensagem e espera resposta
MPI_isend	Passa referência para mensagem de saída e continua
MPI_issend	Passa referência para mensagem de saída e espera até recebedor iniciar
MPI_recv	Recebe uma mensagem; bloqueia se não tem
MPI_irecv	Verifica se existe mensagem, mas sem bloqueio

Algumas primitivas de MPI.

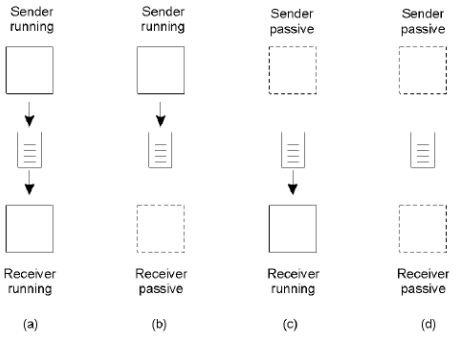
27) MODELO FILA DE MENSAGENS

Sistemas de enfileiramento de mensagens (ou middleware orientado a mensagem) proporcionam suporte extensivo para **comunicação assíncrona persistente**. A essência desses sistemas é que eles oferecem capacidade de armazenamento de médio prazo para mensagens, sem exigir que o remetente ou o receptor estejam ativos durante a transmissão de mensagem.

Uma diferença importante entre a interface Sockets, MPI e Sistemas de Enfileiramento de Mensagens é que estes normalmente visam ao suporte de transferência de mensagens que têm permissão de durar minutos em vez de segundos ou milissegundos.

A ideia básica que fundamenta um sistema de enfileiramento de mensagens é que aplicações se comunicam inserindo mensagens em filas especificas. Essas mensagens são repassadas por uma serie de servidores de comunicação e, a certa altura, entregues ao destinatário, mesmo que ele não esteja em funcionamento quando a mensagem foi enviada.

A mensagem colocada na fila permanece nela ate ser removida. O rementente e o receptor podem executar em completa independência um em relação ao outro.



Quatro combinações para comunicação fracamente acoplada (em relação ao tempo) usando filas. Não há garantias QUANDO ou SE a mensagem será lida pelo receptor.

Na Figura ao lado em (a), o remente e o receptor executam durante toda a transmissão de uma mensagem.

Em (b), somente o rementente está em execução, enquanto o receptor está passivo, isto é, em um estado no qual a entrega da mensagem não é possível. Ainda assim, o remetente pode enviar mensagens.

Em (c), é mostrada a combinação de um rementente passivo e um receptor em execução. Nesse caso, o receptor pode ler mensagens que lhe foram enviadas, mas não é necessário que seus respectivos remetentes estejam também em execução.

Em (d), vemos a situação em que o sistema está armazenado (é possivelmente transmitindo) mensagens mesmo enquanto o remetente e o receptor estão passivos.

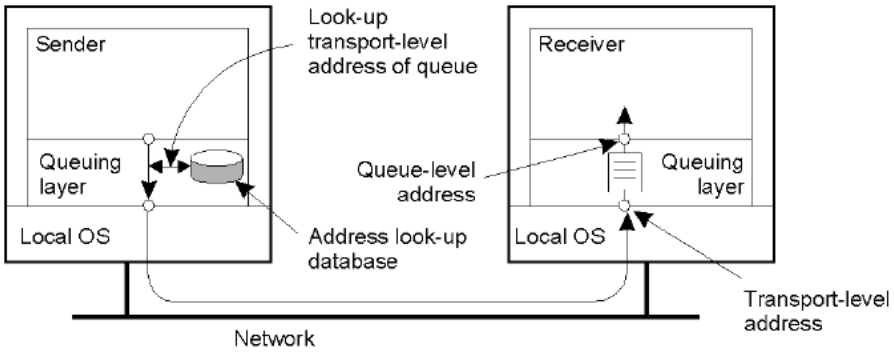
Primitive	Meaning
Put	Anexa uma mensagem em uma fila especificada
Get	Bloqueia até que a fila não esteja vazia e remove a primeira mensagem
Poll	Verifica uma fila determinada e remove mensagem se tiver. Sem bloqueio.
Notify	Instala um tratador para ser chamado quando uma mensagem é colocada na fila.

Interface básica para uma fila em um sistema de fila de mensagens.

ARQUITETURA GERAL DE UM SISTEMA DE FILA DE MENSAGENS

Mensagens só podem ser colocadas em filas locais do rementente, isto é, filas na mesma maquina ou, no máximo, em uma maquina próxima, tal como na mesma LAN, e que possam ser alcançadas de modo eficiente por meio de uma RPC (Filas de fonte). Da mesma, maneira mensagens só podem ser lidas em filas locais.

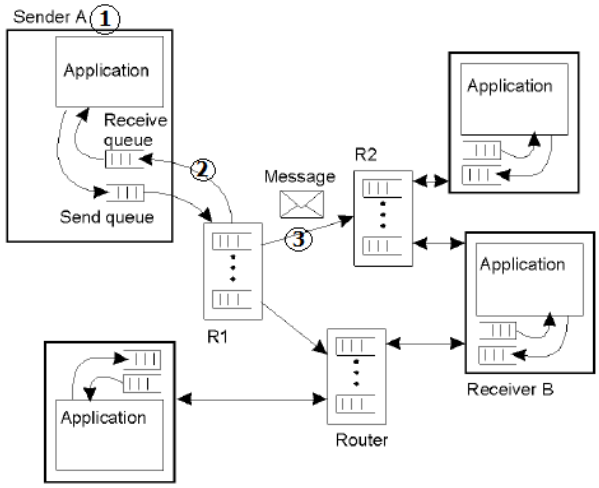
Contudo, uma mensagem colocada em uma fila contem a especificação de uma fila de destino para qual ela deve ser transferida. Cabe ao sistema de enfileiramento de mensagens a responsabilidade de fornecer filas para remententes e receptores e providenciar para que as mensagens sejam transferidas da sua fila de fonte para a sua fila de destino.



Relação entre endereçamento nível de fila e nível de rede.

As filas são distribuídas por varias maquinas. Logo, o Sistema de Enfileiramento precisa manter um mapeamento de filas para localizações de rede. É análogo ao DNS.

O banco de dados da Figura ao lado é um banco de dados distribuído de nomes de filas para localizações de rede.



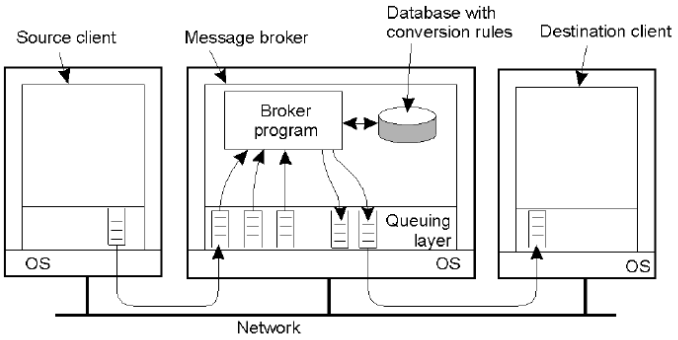
Organização geral de um sistema com roteadores.

Conforme mostra a Figura ao lado. Em 1 A envia uma mensagem para B em sua fila local. Em 2 essa mensagem primeiro é transferida pra o repassador (roteador) mais próximo (R1). Em 3 nesse ponto o R1 sabe o que fazer com a mensagem e a repassa na direção de B.

Desse modo, so os repassadores precisam ser atualizados quando filas são adicionadas ou removidas, enquanto todos os gerenciadores de fila so tem que saber onde está o repassador mais próximo.

Um exemplo de Sistemas de Enfileiramento de mensagens é o WebSphere da IBM.

28) MESSAGE BROKERS



Organização geral de um conversor de mensagem em um sistema de fila de mensagem.

Em Sistemas de Enfileiramento de Mensagens, conversões são manipuladas por nós especiais em uma rede de enfileiramento, conhecidos como **brokers de mensagens (Message Brokers)**.

Um Brokers de mensagens age como um gateway de nível de aplicação em um sistema de enfileiramento de mensagem. Sua principal finalidade é converter as mensagens que chegam de modo que elas sejam entendidas pela aplicação destinatária.

29) COMUNICAÇÃO ORIENTADA A STREAM

Também conhecida como Comunicação Orientada a Fluxo. Trata-se de uma comunicação que depende do tempo como áudio,vídeo, etc.

As facilidades que sistemas distribuídos oferecem para trocar informações dependentes do tempo: áudio e vídeo streams.

Mídia contínua → relação temporal entre os diferentes itens de dados é fundamental para interpretação correta do significado. Considere movimento, uma série de imagens sendo mostradas com espaços uniformes (30-40mseg por imagem).

Mídia discreta → relação temporal não é fundamental para interpretação correta.

Capturar troca de informação dependente de tempo - stream de dados (data stream), seqüência de dados, aplicado tanto a mídia contínua quanto discreta (pipes, TCP/IP).

Transmissão assíncrona → itens no stream é transmitida um após outro sem restrições temporais. Ex. arquivo.

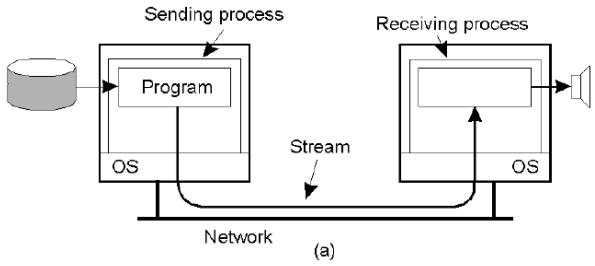
Transmissão síncrona → existe um retardo máximo definido para cada unidade sem importar se a transferência é mais rápida: ex. sensor de temperatura.

Transmissão isócrona → é necessário que unidades sejam transferidas no tempo, sujeito a um máximo e mínimo retardo (jitter limitado): ex. Vídeo

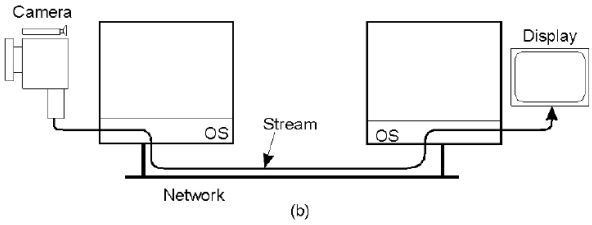
Stream pode ser simples (mono) ou complexo (estéreo, filme).

Após especificação o SD pode alocar recursos (banda, buffer, processamento) para estabelecer um stream que satisfaz os requisitos de Qualidade de Serviço (QoS).Tipos de QoS que podem ser implementados:

- Utilização de um buffer para reduzir variancia de atraso
- Sincronização



Preparação de um stream entre 2 processos através de uma rede.
(Datagrama de melhor esforço: IP) = (Problema de QoS)



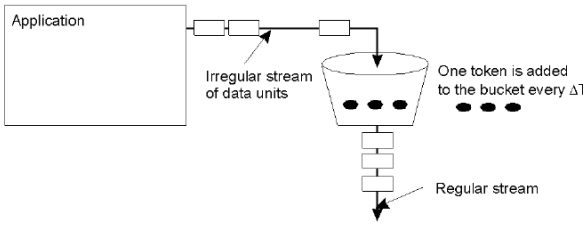
Preparação de um stream direto entre 2 dispositivos através de uma rede.

A Figura (a) mostra uma arquitetura cliente-servidor geral para suportar fluxos de dados contínuos de multimídia armazenados por uma rede. Essa arquitetura revela varias questões importantes como os dados de multimídia (vídeo e áudio) precisarão ser comprimidos de modo a reduzir o armazenamento, controle da qualidade da transmissão e questões de sincronização.

A Figura (b) mostra um fluxo de dados transmitidos ao vivo. Os dados capturados em tempo real são enviados a receptores pela rede. A principal diferença entre o armazenado na rede e o ao vivo é que a transmissão de fluxo de dados ao vivo oferece menos oportunidades de sintonizar um fluxo.

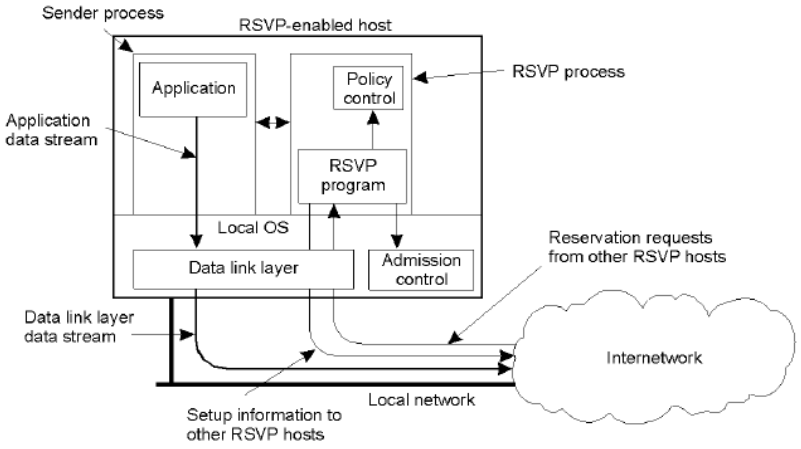
Características da Entrada	Serviço necessário
• tamanho máximo unidade (bytes) • taxa de tokens (bytes/sec) • tamanho recipiente (bytes) • taxa máxima de transmisão (bytes/sec)	•Sensibilidade de perda (bytes) •intervalo de perda (μsec) •perda consecutiva (data units) •retardo mínimo notado (μsec) •jitter máximo (μsec) •Qualidade da garantia

Especificação de fluxo.



O princípio do algoritmo token bucket.

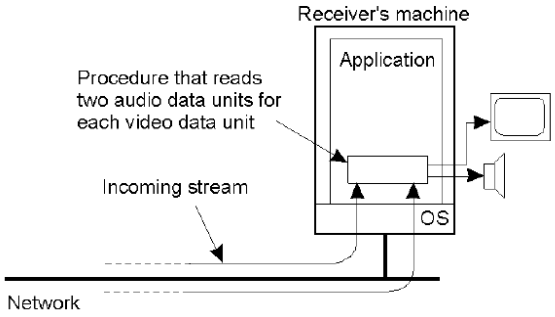
Estabelecendo um Stream



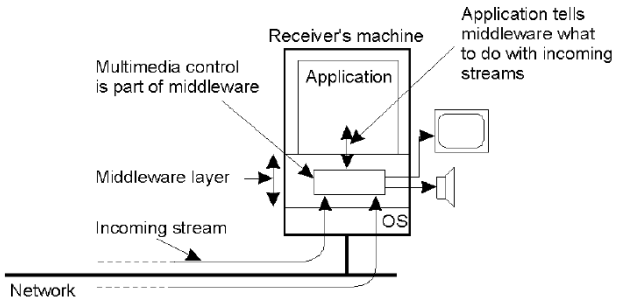
Organização básica de protocolo para reserva de recursos em um sistema distribuído - RSVP

Mecanismos de Sincronização podem ser vistos em diferentes niveis de abstração conforme mostram as Figuras a seguir.

Synchronization Mechanisms (1)



Synchronization Mechanisms (2)



Na Figura Synchronization Mechanisms (1) mostra que no nivel mais baixo, a sincronizacao é feita explicitamente por operacoes sobre unidades de dados de fluxos simples. Em essencia, há um processo que simplesmente executa operacoes de leitura e escrita em varios fluxos simples, assegurando que essas operacoes obedecem as restricoes especificadas de temporizacao e sincronizacao.

A desvantagem dessa abordagem é que a aplicacao fica totalmente responsavel por implementar a sincronizacao, embora so tenha a disposicao facilidades de baixo nivel.

A Figura Synchronization Mechanisms (2) mostra uma abordagem mais apropriada que é oferecer a uma aplicacao uma interface que lhe permita um controle mais facil de fluxos e dispositivos. Esse é tipico de muito sistemas de middleware de multimidia.