Sistemas Distribuídos

Sistemas Operacionais

Prof. Pedro Ramos
pramos.costar@gmail.com

Pontifícia Universidade Católica de Minas Gerais
ICEI - Departamento de Ciência da Computação

Resumo do curso

Cobrimos todos os componentes fundamentais do sistema operacional:

- Arquitetura e interações do sistema operacional
- Processos e threads
- Sincronização e deadlock
- Programação de processos
- Gerenciamento de memória
- Sistemas de arquivos e E/S

Sistemas distribuídos

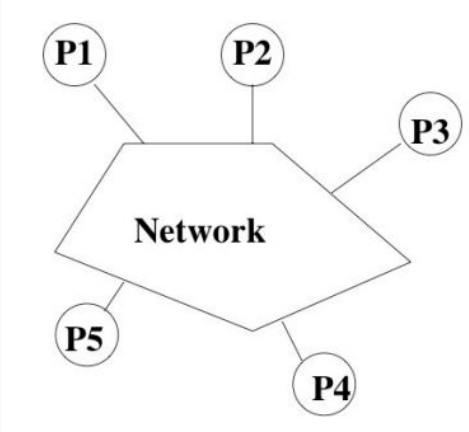
- Noções básicas de rede
- Serviços distribuídos (e-mail, www, telnet)
- Sistemas operacionais distribuídos
- Sistemas de arquivos distribuídos

Sistemas distribuídos

Sistema distribuído: um conjunto de **processadores** fisicamente **separados**, **conectados** por um ou mais **links de comunicação**

- Quase todos os sistemas atuais são distribuídos de alguma forma.

  – *E-mail, servidores de arquivos, impressoras de rede, backup remoto, rede mundial de computadores*



Sistemas paralelos VS sistemas distribuídos

Sistemas **fortemente acoplados**: "*processamento paralelo*"

- Os processadores **compartilham relógio, memória** e executam **um só sistema operacional**
- Comunicação frequente

Sistemas com **acoplamento frouxo**: "*computação distribuída*"

- **Cada processador tem sua própria memória**
- Cada processador executa um **sistema operacional independente**
- A **comunicação** deve ser **menos frequente**

Vantagens

Compartilhamento de recursos:

– Os recursos não precisam ser **replicados** em cada processador (por exemplo,arquivos compartilhados)
– Recursos caros (**escassos**) podem ser compartilhados (por exemplo, impressoras)
– Cada processador pode apresentar o **mesmo** **ambiente** ao usuário (por exemplo, mantendo os arquivos em um servidor de arquivos)

Aumento da velocidade computacional:

– n processadores potencialmente lhe dão n vezes o **poder computacional**
– Os problemas devem ser **decompostos** em **subproblemas**
– É necessário haver coordenação e comunicação entre os processos de cooperação (**sincronização**, troca de resultados).

Vantagens

Confiabilidade:

- A replicação de recursos gera **tolerância** a **falhas**.
- Por exemplo, se um nó falhar, o usuário poderá trabalhar em outro.
- O desempenho será prejudicado, mas o sistema permanecerá operacional (*operational*).
- No entanto, se algum componente do sistema for **centralizado**, poderá haver um **único** **ponto** de falha
- Exemplo: Se uma estação de trabalho do CRC falhar, você poderá usar outra estação de trabalho. Se o servidor de arquivos inteirodo CRC falhar, nenhuma das estações de trabalho será útil.

Comunicação:

- Usuários/processos em ***sistemas diferentes podem se comunicar***.
- Por exemplo, correio, sistemas de processamento de transações, como companhias aéreas e bancos, WWW.

Sistemas distribuídos

• Os ambientes de trabalho modernos são distribuídos=> sistemas operacionais precisam ser distribuídos

• O que precisamos considerar ao criar esses sistemas?

- **Comunicação e redes**
- **Transparência (quão visível é a distribuição?)**
- **Segurança**
- **Confiabilidade**
- **Desempenho e escalabilidade**
- **Modelos de programação**

Sistemas distribuídos

O que fica mais difícil quando passamos de um sistema autônomo para um ambiente distribuído?

- compartilhamento de recursos
- tempo (por exemplo, sincronização)
- seções críticas
- detecção e recuperação de deadlock
- recuperação de falhas

Redes

Em geral, as redes se preocupam em fornecer uma passagem de mensagens eficiente, correta e robusta entre dois **nós distintos (ou nodos)**.

A rede local (LAN) geralmente conecta nodos em um único edifício e precisa ser rápida e confiável (por exemplo, protocolo *Ethernet*).

- Mídia: par trançado, cabo coaxial, fibra óptica
- Largura de banda típica: 10-100-1000 Mb/s (10Gb/s a 400Gb/s já disponível em datacenters)

A **Wide Area Network (WAN)** conecta nodos em todo o estado, país ou planeta.

- As WANs são normalmente mais lentas e menos confiáveis do que as LANs (por exemplo, a Internet).
- Mídia: satélite, roteadores, cabos residenciais.
- Largura de banda típica: Starlink (250 Mb/s), 5G (10 Gb/s), Wi-Fi 7.0 (40 Gb/s), Bluetooth (2 Mb/s)

Princípios de comunicação em rede

• Os dados enviados para a rede são divididos em "**pacotes**", a *unidade básica* de *transmissão* da rede. Os pacotes são enviados pela rede.
• Os ***computadores nos pontos de comutação*** *controlam o fluxo de pacotes*.

Analogia:
carros / estrada / polícia
pacotes / rede / computador

• Os recursos compartilhados podem levar à **contenção** (congestionamentos).

Analogia:
- Nodo compartilhado - Shopping center
- Link compartilhado - Ponte

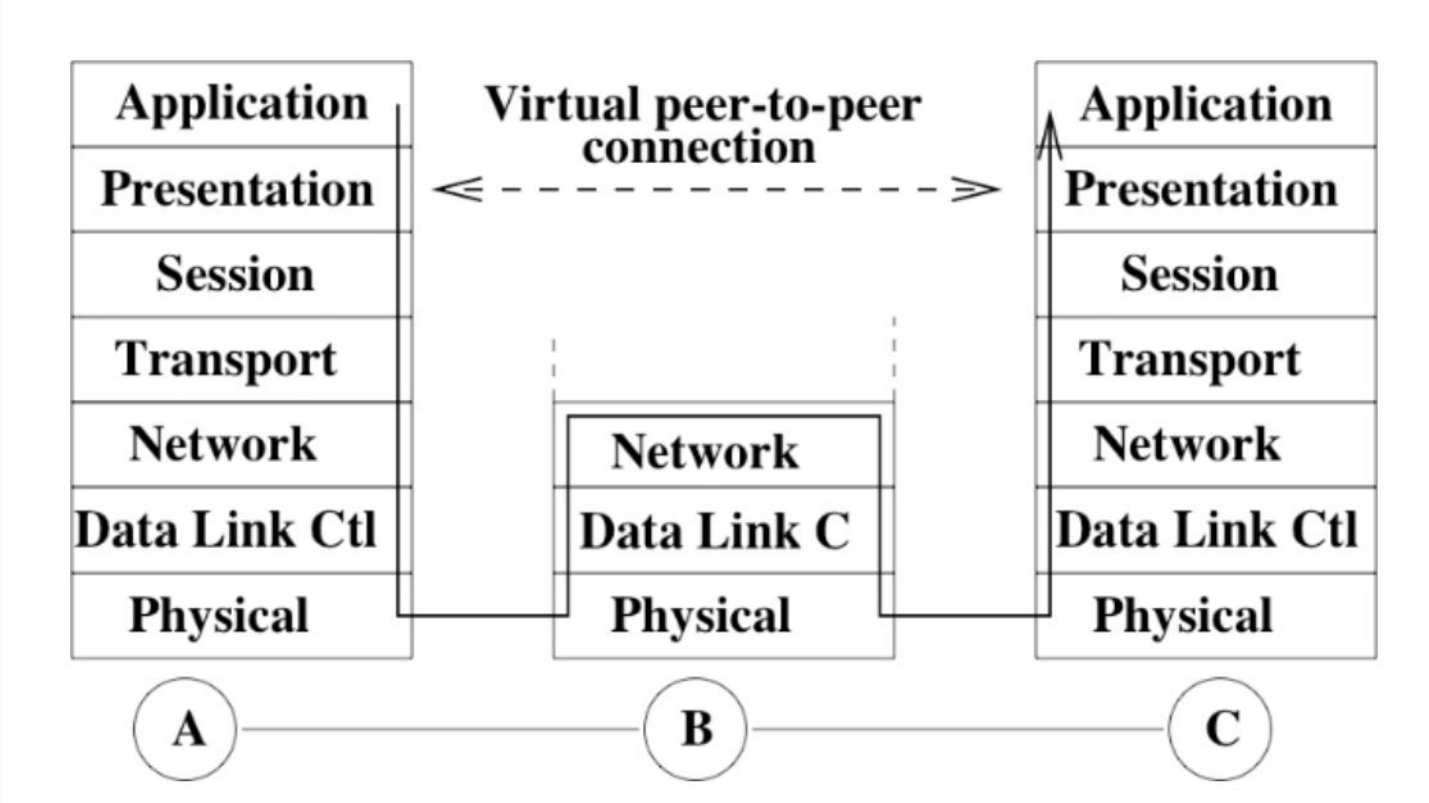
Protocolos de Comunicação

Protocolo: um conjunto de regras de comunicação que são acordadas por todas as partes

- Pilha de protocolos: o software de rede é estruturado em camadas
  - **Cada camada N fornece um serviço para a camada N+1, usando seus próprios procedimentos da camada N e a interface para a camada N-1.**

  - Exemplo: Organização Internacional de Padrões/Interconexão de Sistemas Abertos (ISO/OSI)

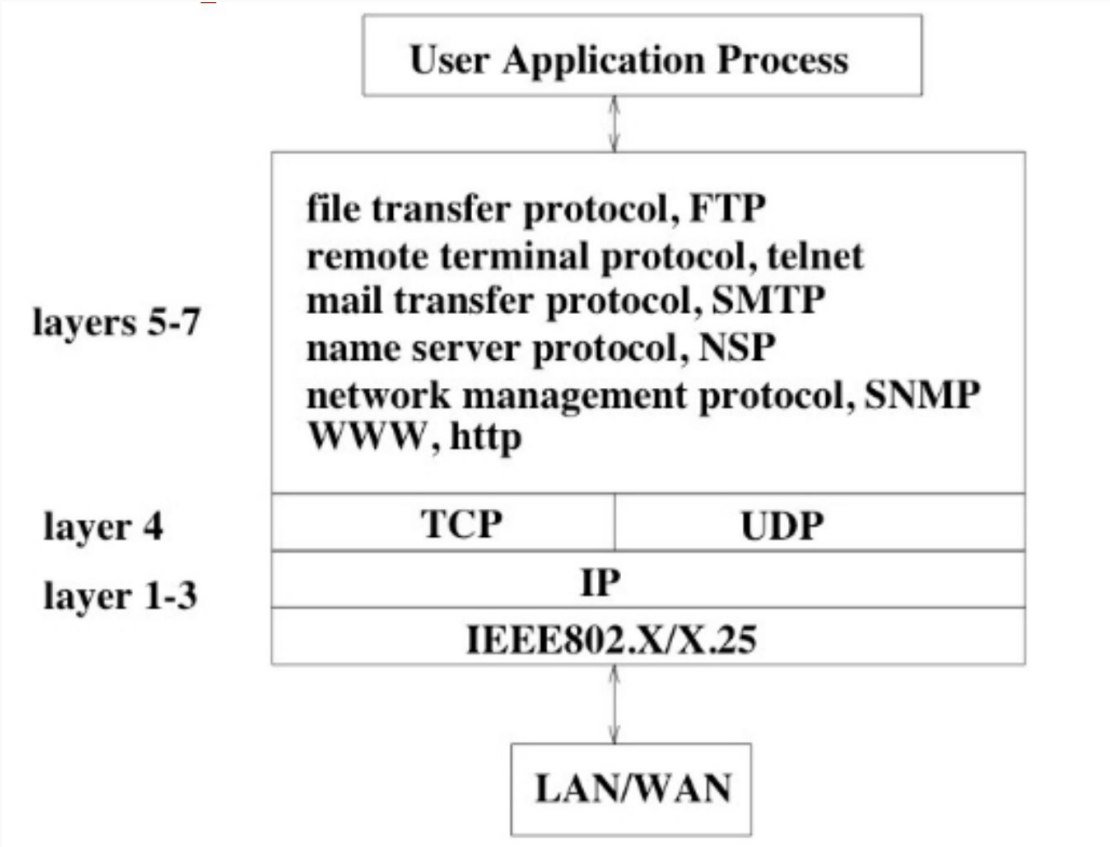
Protocolos


Application
Presentation
Session
Transport
Network
Data Link Ctl
Physical
Virtual peer-to-peer connection
Network
Data Link C
Physical
Application
Presentation
Session
Transport
Network
Data Link Ctl
Physical
A
B
C

Pilha de protocolos de rede ISO

Camada de aplicativos: aplicativos que usam a rede, por exemplo, e-mail, http, ftp, p2p, www, etc. Fornecem uma interface de usuário.
Camada de apresentação: conversão de formato de dados, por exemplo, formato inteiro big/little endian. Compatibilidade com outros SOs)
Camada de sessão: implementa a estratégia de comunicação, como RPC. Fornecida por bibliotecas.
Camada de transporte: comunicação confiável de ponta a ponta entre qualquer conjunto de nós. Fornecida pelo sistema operacional.
Camada de rede: roteamento e controle de congestionamento. Geralmente implementado no sistema operacional.
Camada de controle de link de dados (enlace): comunicação ponto a ponto confiável de pacotes em um canal não confiável. Às vezes implementada em hardware, às vezes em software.
Camada física: sinalização elétrica/óptica em um "fio". Lida com problemas de tempo. Implementado em hardware.

Pilha de protocolos de rede ISO



Pilha de protocolos de rede TCP/IP

A maioria dos sites da Internet usa ***TCP/IP - Protocolo de Controle de Transmissão/Protocolo de Internet.***

– Ele tem **menos camadas** do que o ISO para aumentar a eficiência.

– Consiste em um **conjunto** de **protocolos**: **UDP**, **TCP**, **IP**...

– O **TCP** é um protocolo **confiável**: os pacotes são recebidos na ordem em que são enviados.

– **UDP** (protocolo de datagrama do usuário), um protocolo **não confiável** (sem garantia de entrega).

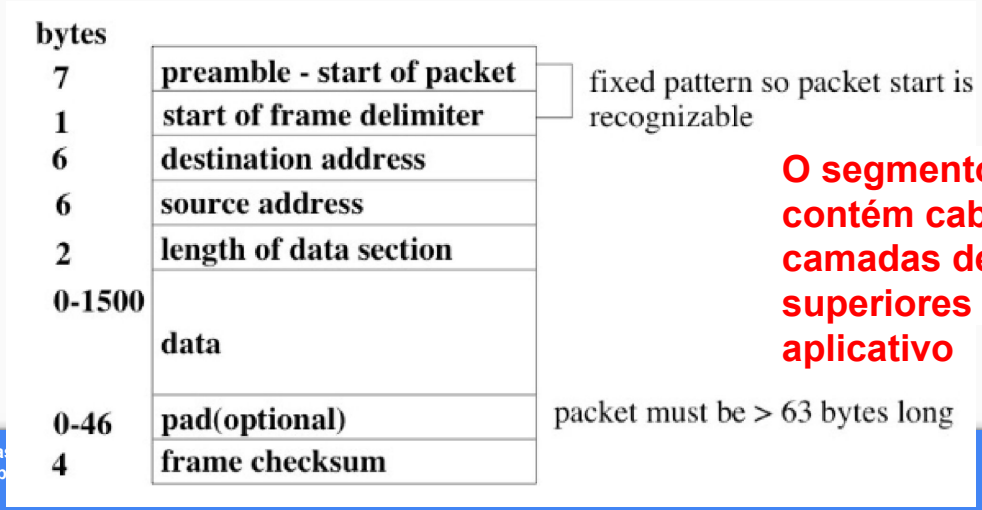
Pacote

Cada mensagem é **dividida** em **pacotes**.

– Cada **pacote** contém **todas** as **informações** necessárias para **recriar** a **mensagem** **original**.

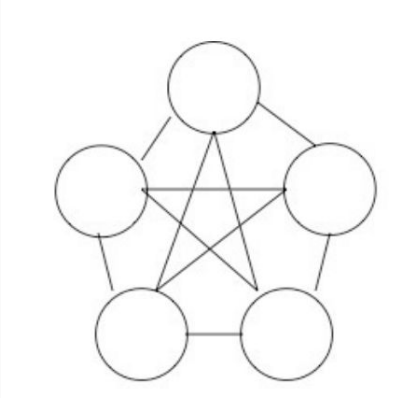
– Por exemplo, os pacotes podem chegar fora de ordem e o nó de destino deve ser capaz de colocá-los em ordem novamente.

– Conteúdo do pacote Ethernet:



O segmento de dados do pacote contém cabeçalhos para camadas de protocolo superiores e dados reais do aplicativo

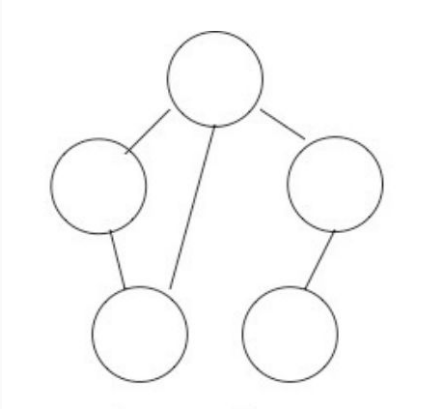
Topologias de rede ponto a ponto



Totalmente conectado: todos os nodos estão conectados a todos os outros nodos

- Cada **mensagem leva apenas um único "salto"**, ou seja, vai diretamente para o destino sem passar por nenhum outro nodo
- A falha de um nodo não afeta a comunicação entre os outros nodos
- Caro, especialmente com muitos nodos, não é prático para WANs

Topologias de rede ponto a ponto



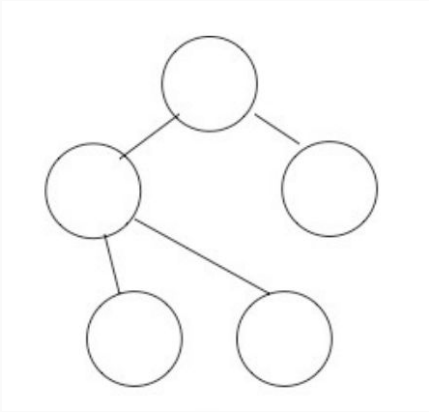
Parcialmente conectado: links entre alguns, mas não todos os nodos

- **Menos caro, mas menos tolerante a falhas**. Uma única falha pode dividir a rede.
- **O envio de uma mensagem a um nodo pode ter que passar por vários outros nodo**

=> precisam de **algoritmos de roteamento**.

- As **WANs** normalmente usam essa estrutura

Topologias de rede ponto a ponto



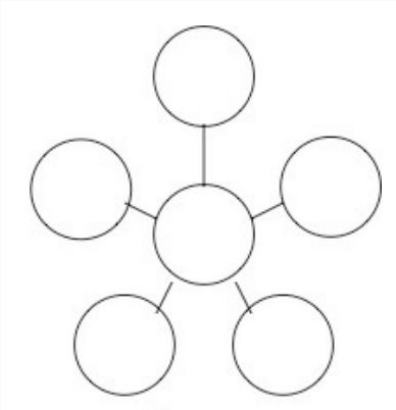
Estrutura em árvore: hierarquia de rede

– Todas as mensagens **entre descendentes diretos são rápidas**, mas as mensagens **entre "primos" devem subir até um ancestral comum** e depois descer novamente.

– Algumas redes corporativas usam essa topologia, pois ela corresponde a uma visão hierárquica do mundo.

– **Não tolera falhas.** Se algum nó interno falhar, a rede será dividida.

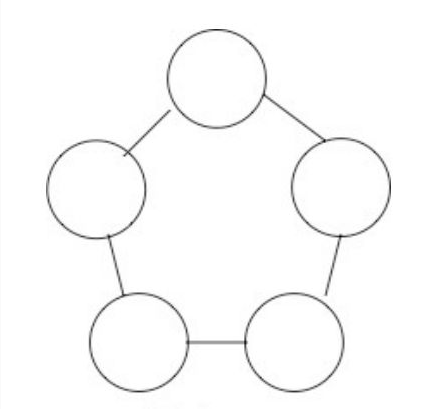
Topologias de rede ponto a ponto



Estrela: todos os nós se conectam a um único nó centralizado

- O **nodo central** geralmente é dedicado ao **tráfego de rede.**
- Cada mensagem leva apenas dois saltos.
- **Se uma peça de hardware falhar, toda a rede será desconectada**.
- Barato e, às vezes, usado para **LAN**

Topologias de rede ponto a ponto



Anel unidirecional - os nós **só podem enviar em uma direção.**

– Considerando n nós, a mensagem pode precisar percorrer ***n-1 saltos.***

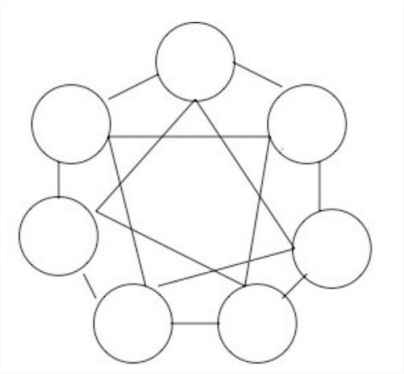
– Barato, mas uma falha divide a rede.

Anel bidirecional - os nós podem enviar em **qualquer direção.**

– Com n nós, uma mensagem precisa percorrer no máximo ***n/2 saltos***.

– Barato, tolera uma única falha aumentando os saltos das mensagens. Duas falhas dividem a rede.

Topologias de redes em anel

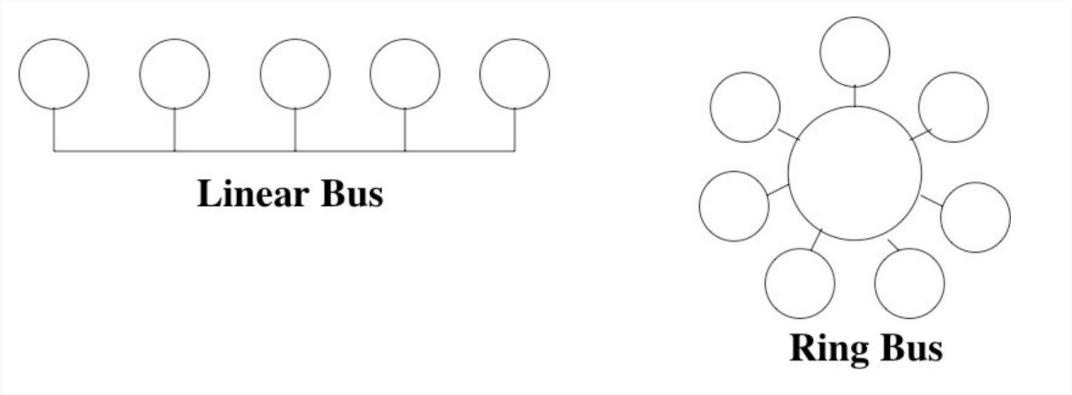


Nós de anel duplamente conectados:
conectados a vizinhos e a um vizinho distante

- Uma mensagem leva no máximo ***n/4 saltos***.

- Mais caro, ***mas mais tolerante a falhas***.

Topologias de rede de barramento



Os nós de barramento se conectam a uma rede comum

Barramento linear - link (enlace) único compartilhado
– Os nós se conectam diretamente uns aos outros usando a tecnologia de barramento de multiacesso.
– **Barato (linear no número de nodos) e tolerante a falhas de nodos.**
– A LAN ***Ethernet*** usa essa estrutura.
Barramento em anel - link circular único compartilhado
– A mesma tecnologia e as mesmas vantagens e desvantagens de um barramento linear.

Compartilhamento de recursos

Há muitos mecanismos para compartilhar recursos (hardware, software, dados).

- **Migração** de **dados**: mover os **dados**
- **Migração** de **computação**: mover a **computação** para os dados
- **Migração** de **trabalho**: mover o **trabalho** (computação e dados) ou parte dele

=> A troca fundamental no compartilhamento de recursos é **concluir as instruções do usuário da forma mais rápida** e econômica possível.

(Rápido e econômico geralmente são incompatíveis).

Modelo cliente/servidor

Um dos modelos mais comuns para estruturar a computação distribuída é usar o **paradigma cliente/servidor**.

- Um servidor é um processo ou **conjunto de processos que fornecem um serviço**, por exemplo, serviço de **nomes**, serviço de **arquivos**, serviço de **banco de dados** etc.
- O servidor pode existir em um ou mais nodos.
- Um cliente é um programa que usa o serviço.
- Um cliente primeiro se vincula ao servidor, ou seja **localiza-o na rede e estabelece uma conexão**.
- Em seguida, o **cliente envia ao servidor uma solicitação para executar alguma ação**. O servidor envia de volta uma resposta.

    - *A RPC é uma maneira comum de implementar essa estrutura.*

Chamada de procedimento remoto (RPC)

Ideia básica do RPC:

- Os **servidores exportam** procedimentos (**funções**) para serem chamados por um conjunto de clientes.

- Para usar o servidor, o **cliente faz** uma **chamada** de procedimento (**função**).

- O **sistema operacional gerencia** a **comunicação**.

Chamada de procedimento remoto: Problemas de implementação

Para cada procedimento no qual desejamos oferecer suporte a RPC:

O mecanismo RPC usa a assinatura do procedimento (número e tipo de argumentos e valor de retorno)

– *para **gerar um stub de cliente** que agrupa os argumentos de RPC e os envia ao servidor, e*

– *para **gerar o stub do servidor** que descompacta a mensagem e faz a chamada do procedimento.*

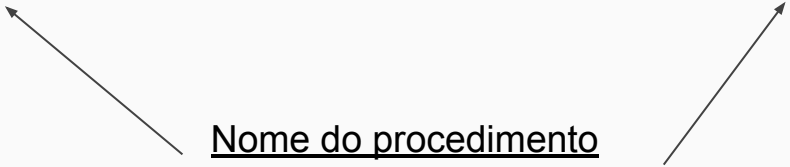
Chamada de procedimento remoto: Problemas de implementação

```
Stub do cliente:

criar mensagem
enviar mensagem
aguardar resposta
desempacotar
resposta
retornar resultado
```

```
Servidor Stub:

criar loop de threads
    esperar por um comando
    descompactar parâmetros de solicitação
    chamar procedimento com thread
    criar resposta com resultado(s)
    enviar resposta
loop final
```



Nome do procedimento
Parâmetros
Resultado
Endereço de retorno

Chamada de procedimento remoto

- **Como o cliente sabe qual é a porta correta?**
  - A associação pode ser **estática**, fixada no momento da compilação.
  - Ou a vinculação pode ser **dinâmica**, fixada em tempo de execução.

- **Na maioria dos sistemas RPC, a vinculação dinâmica é realizada usando um serviço de nomes.**
  - Quando o servidor é iniciado, ele exporta sua interface e se identifica em um **servidor de nomes de rede**
  - O cliente, antes de fazer qualquer chamada, ***solicita ao serviço de nomes a localização de um servidor cujo nome ele conhece e***, em seguida, estabelece uma conexão com o servidor

DNS

Exemplo: Invocação de método remoto (RMI) em Java

- O Java fornece as seguintes classes/interfaces:
  - **Naming**: classe que fornece as chamadas para se comunicar com o registro de objetos remotos
  - **public static void bind(String name, Remote obj)** - Vincula um servidor a um nome.
  - **public static Remote lookup(String name)** - Retorna o objeto do servidor que corresponde a um nome.

- **UnicastRemoteObject**: suporta referências a objetos remotos não replicados usando TCP, exporta a interface automaticamente quando o objeto do servidor é construído

- O Java oferece as seguintes ferramentas:
  - servidor de nomes no lado do servidor **rmiregistry**
  - **rmic**: dada a interface do servidor, gera stubs de cliente e servidor que criam e interpretam pacotes

Exemplo: Server em Java

Servidor

– Define uma interface que lista as assinaturas dos métodos que o servidor atenderá

– Implementa cada um dos métodos da interface

– Programa principal do servidor:

- Cria um ou mais objetos de servidor - chamada normal do construtor quando o objeto que está sendo construído é uma subclasse de RemoteObject
- Registra os objetos no registro de objetos remoto

Cliente

– Procura o servidor no registro de objetos remotos

– Usa a sintaxe normal de chamada de método para métodos remotos

– Deve tratar a RemoteException

Exemplo: Server em Java

Declare os métodos que o servidor fornece:

```
pacote examples.hello;

// Todos os servidores devem estender a interface Remote.
interface pública Hello extends java.rmi.Remote {

// Qualquer método remoto pode lançar RemoteException.
// Indica falha na rede.
    String sayHello() // é um caso de java.rmi.RemoteException;
}
```

Exemplo: Servidor Hello World

```
package examples.hello;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {

    public HelloImpl() throws RemoteException {
        // O construtor da superclasse exporta a interface e obtém uma porta
        super();
    }

    public String sayHello() throws RemoteException {
        // Esse é o "serviço" fornecido.
        return "Hello World!";
    }
```

Exemplo: Servidor Hello World

```
// Continuação da classe:

public static void main(String args[]) {
        // Criar e instalar um gerenciador de segurança
        System.setSecurityManager(new RMISecurityManager());

        // Construir o objeto do servidor. HelloImpl
        obj = new HelloImpl();

        // Registre o servidor com o servidor de nomes.
        Naming.rebind("//myhost/HelloServer", obj);
    }
}
```

Exemplo: Cliente Hello World

```
pacote examples.hello;
importar java.awt.*;
importar java.rmi.*;
public class HelloApplet extends java.applet.Applet {
    String message = "";

    // O método init inicia a execução do applet no cliente
    // máquina que está visualizando a página da Web que contém a referência
    // para o applet. public
    void init() {
        try {

            // Procura o servidor usando o servidor de nomes no host que
            // de onde veio o applet.
            Hello obj= (Hello)Naming.lookup(
                            "//"+ getCodeBase().getHost()+ "/HelloServer");


...
```

Exemplo: Cliente Hello World: Continuação

```
...

            // Chama o método sayHello no objeto remoto. message =
            obj.sayHello();
        } catch (RemoteException e) {
            System.out.println("HelloApplet RemoteException caught");
        }
    }

    public void paint(Graphics g) {
        // O applet escreverá a string retornada pelo método remoto
        // chamada na tela.
        g.drawString(message, 25, 50);
    }
}
```

Resumo

- Praticamente todos os sistemas de computador contêm componentes distribuídos

- As redes os conectam

- As redes fazem concessões entre velocidade, confiabilidade e custo

PERGUNTAS?

REFERÊNCIAS

- **TANENBAUM, Andrew.** Sistemas operacionais modernos.
- **SILBERSCHATZ, Abraham et al.** Fundamentos de sistemas operacionais: princípios básicos.
- **MACHADO, Francis; MAIA, Luiz Paulo.** Arquitetura de Sistemas Operacionais.
- **CARISSIMI, Alexandre et al.** Sistemas operacionais.