SISTEMAS DISTRIBUÍDOS

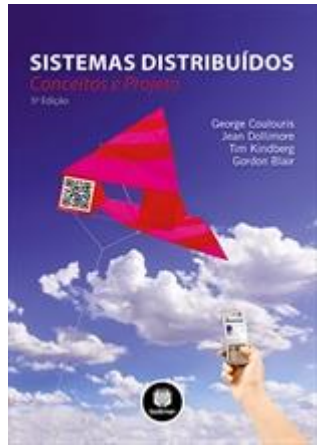
Professor: Johnatan Oliveira

Material adaptado do professor Renato

SUMÁRIO

- Exercícios
- Processos em Sistemas Distribuídos e Comunicação entre Processos
  - Introdução
  - Processos
  - Threads
  - Características da comunicação entre processos
  - A API para protocolos na Internet
- Trabalho prático

Cap 4



O QUE VIMOS...

- Um modelo de arquitetura de um sistema distribuído envolve o posicionamento de suas partes e os relacionamentos entre elas
- Modelos de arquitetura de SD:
  - Cliente-servidor;
  - Peer-to-peer;
  - Múltiplos servidores;
  - Servidores de proxy e cache; Códigos móveis;
  - *Camada lógicas, Camadas físicas, Thin clients, proxy, ...*

- Requisitos no projeto de arquiteturas distribuídas:
  - Desempenho;
  - Qualidade de serviço;
  - Uso de caching e replicação

O QUE VIMOS...

- Repertório de aspectos usados em modelos de SD:
  - Aspectos de modelos de interação:
    - Assíncrono
    - Síncrono
  - Aspectos de modelos de falhas
    - Nos processos; nos canais de comunicação
    - De omissão, de tempo, arbitrárias
  - Aspectos de modelos de segurança
    - Modelos de ameaças
    - Criptografia, autenticação, autorização, canal seguro

EXERCÍCIOS

- 1) Ano: 2014. Banca: IADES. Órgão: FUNPRESP
- Cargo: Analista Técnico de Tecnologia da Informação - Área Suporte
- O conceito de transparência pode ser aplicado a diversos aspectos de um sistema distribuído. Assinale a alternativa correta quanto ao tipo de transparência e a respectiva descrição.
- a) Acesso –mostra o lugar onde um recurso está localizado.
- b) Replicação –oculta que um recurso é replicado
- c) Relocação –oculta que um recurso não pode ser movido para outra localização estando fora de uso.
- d) Migração –mostra que um recurso pode ser movido para outra localização.
- e) Falha –mostra a falha e a recuperação de um recurso.

EXERCÍCIOS

- 2) Um mecanismo de busca é um servidor web que responde aos pedidos do cliente para pesquisar em seus índices armazenados e (concomitantemente) executa várias tarefas de *web crawling* para construir e atualizar esses índices.
- Quais são os requisitos de sincronização entre essas atividades concomitantes?

EXERCÍCIOS

- 3) Considere uma empresa de aluguel de carros hipotética e esboce uma solução de três camadas físicas para seu serviço distribuído de aluguel de carros.
- Use sua resposta para ilustrar vantagens e desvantagens de uma solução em três camadas físicas, considerando problemas como desempenho, mudança de escala, tratamento de falhas e manutenção do software com o passar do tempo.

EXERCÍCIOS

- 4) Considere um servidor simples que executa pedidos do cliente sem acessar outros servidores.
- Explique por que geralmente não é possível estabelecer um limite para o tempo gasto por tal servidor para responder ao pedido de um cliente.
- O que precisaria ser feito para tornar o servidor capaz de executar pedidos dentro de um tempo limitado? Essa é uma opção prática?

EXERCÍCIOS

- 5) Suponha que a leitura de um disco possa, às vezes, ler valores diferentes dos gravados.
- Cite os tipos de falha exibidos por uma leitura de disco.
- Sugira como essa falha pode ser mascarada para produzir uma forma de falha benigna diferente.
- Agora, sugira como se faz para mascarar a falha benigna.

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- Introdução
  - Os computadores executam várias operações ao mesmo tempo:
    - Por exemplo:
      - compilar um programa;
      - enviar um arquivo para a impressora;
      - exibir uma página Web;
      - reproduzir músicas; e
      - receber mensagens de correio eletrônico;
  - Os **processos** permitem que os sistemas executem e monitorem atividades simultâneas;
  - Os processos transitam entre estados de processo;
  - Os sistemas operacionais executam operações por meio de processos, como criar, destruir, suspender, retomar e acordar.

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- **Processos**
  - **Informações necessárias para execução de processo:**

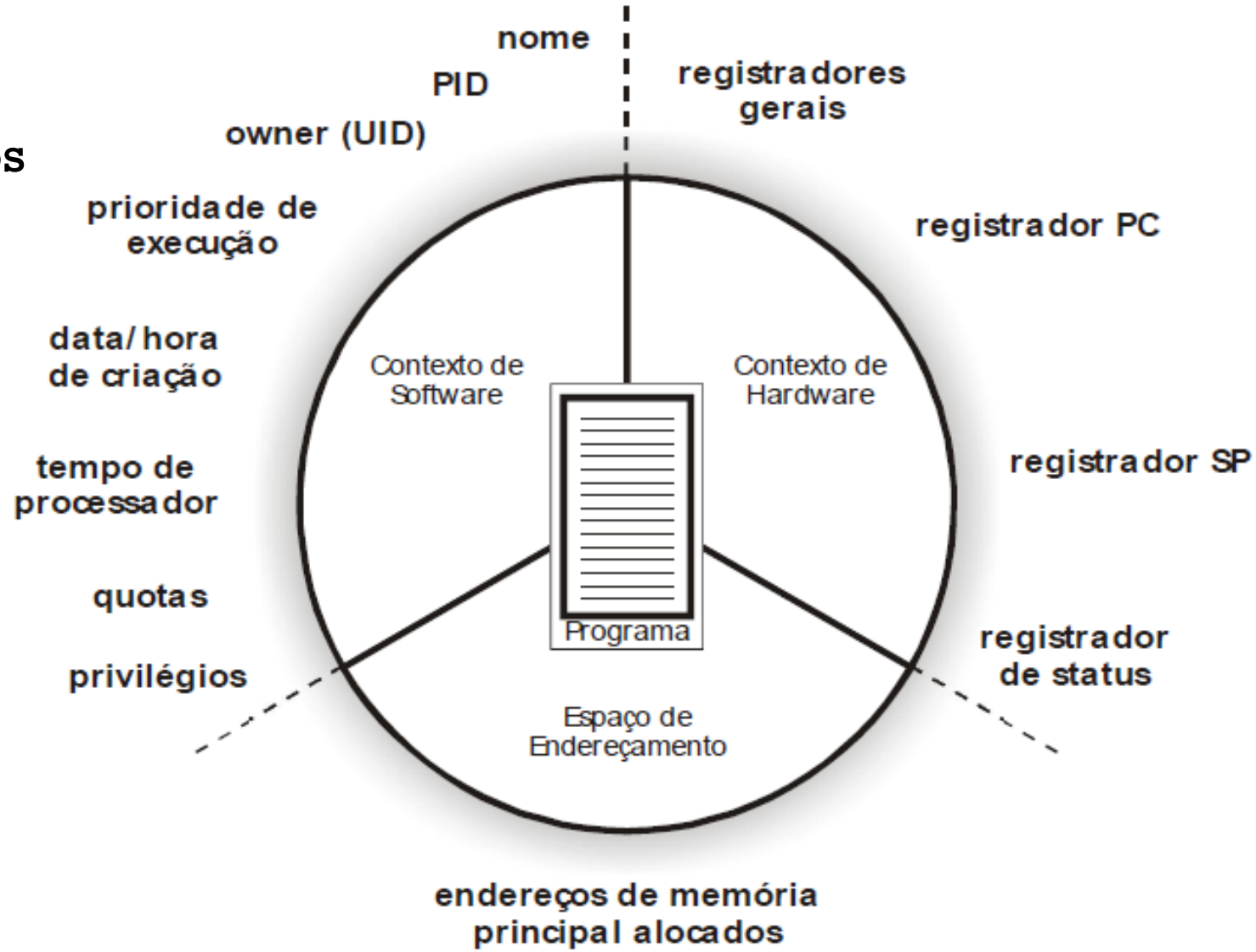
  - **Contexto de hardware**: estado do hardware no momento em que o processo é interrompido para ceder lugar a um outro processo em execução (mudança de contexto);
    - Ex: registradores (pipeline)

  - **Contexto de software**: características e limites dos recursos que podem ser alocados pelo processo;
    - Ex: identificação do processo (PID), usuário "dono" do processo, tamanho máximo em RAM que o processo pode alocar, tamanho máximo do buffer para operações de I/O.

  - **Espaço de endereçamento**: endereço da memória principal alocado para armazenamento de instruções e os dados utilizados pelo processo;

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- Processos



PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- **Processos**
  - **Único fluxo de execução?**
    - Um servidor de arquivos deve esperar por requisições feitas ao disco.
      - O fluxo de execução que fez a requisição é bloqueado aguardando a resposta. (PERDA DE DESEMPENHO);

  - **Solução: Vários Fluxos de Execução**
    - Se o servidor de arquivos é implementado usando diferentes fluxos de execução, outras requisições de clientes podem ser processadas, enquanto o primeiro fluxo aguarda a resposta do disco.
      - MELHOR VAZÃO (*THROUGHPUT*) E GANHO DE DESEMPENHO;

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- Em alguns casos é desejável haver diversos fluxos de execução compartilhando um único espaço de endereçamento, ou seja, mesma região de memória

- ***Threads***
  - Partes de um processo que compartilham mesmo espaço de endereçamento;
  - Sub-rotina de um programa executado paralelamente ao programa chamador (execução concorrente de sub-rotinas);

  - Mais uma definição:
    - *Enquanto processos permitem que o sistema operacional execute mais de uma aplicação ao mesmo tempo, as threads permitem que a aplicação execute mais de um método ao mesmo tempo.*

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- ***Threads***

  - Cada um dos fluxos de execução de um processo é chamado de ***thread***;
  - *Threads* podem ser vistas como **mini-processos**;
  - Cada *thread* executa sua **própria porção de código**;
  - *Threads* compartilham a CPU do mesmo modo que diferentes processos (*timesharing*);
    - Em sistema de threads em geral mantém a mínima informação que permita à CPU ser compartilhada por vários threads

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- **Implementação de Threads em Sistemas Distribuídos**
  - Importante propriedade de threads é que eles podem proporcionar um meio conveniente de permitir chamadas bloqueantes de sistema sem bloquear o processo inteiro;

  - Threads são particularmente atraentes para utilização em **sistemas distribuídos**
    - Facilitam muito expressar comunicação na forma de manter múltiplas conexões lógicas ao mesmo tempo;

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- **Clientes Multithreads**

  - Sistemas distribuídos que operam em redes de longa distância
    - escondem longos tempos de propagação de mensagens entre processos;
    -
  - A maneira de ocultar latências de comunicação é iniciar a comunicação e imediatamente prosseguir com outra atividade;

  - Exemplo: Clientes Multithreads – Browsers Web
    - Documento Web consiste em: texto, imagens, ícones, etc.
    - A cada elemento, browser estabelece uma conexão TCP/IP, para ler os dados e passar ao monitor do usuário;
    - No entanto temos algumas operações bloqueadoras: estabelecimento da conexão, leitura de dados;

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- **Clientes Multithreads**
  - Browsers começam a exibir dados a medida em que novas informações chegam;
  - Enquanto o texto está sendo disponibilizado para o usuário, incluindo as facilidades de rolamento, p.ex., o browser continua buscando outros arquivos, como imagens;
  - **Vantagem**: usuário não precisa esperar até que todos os componentes sejam buscados;

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

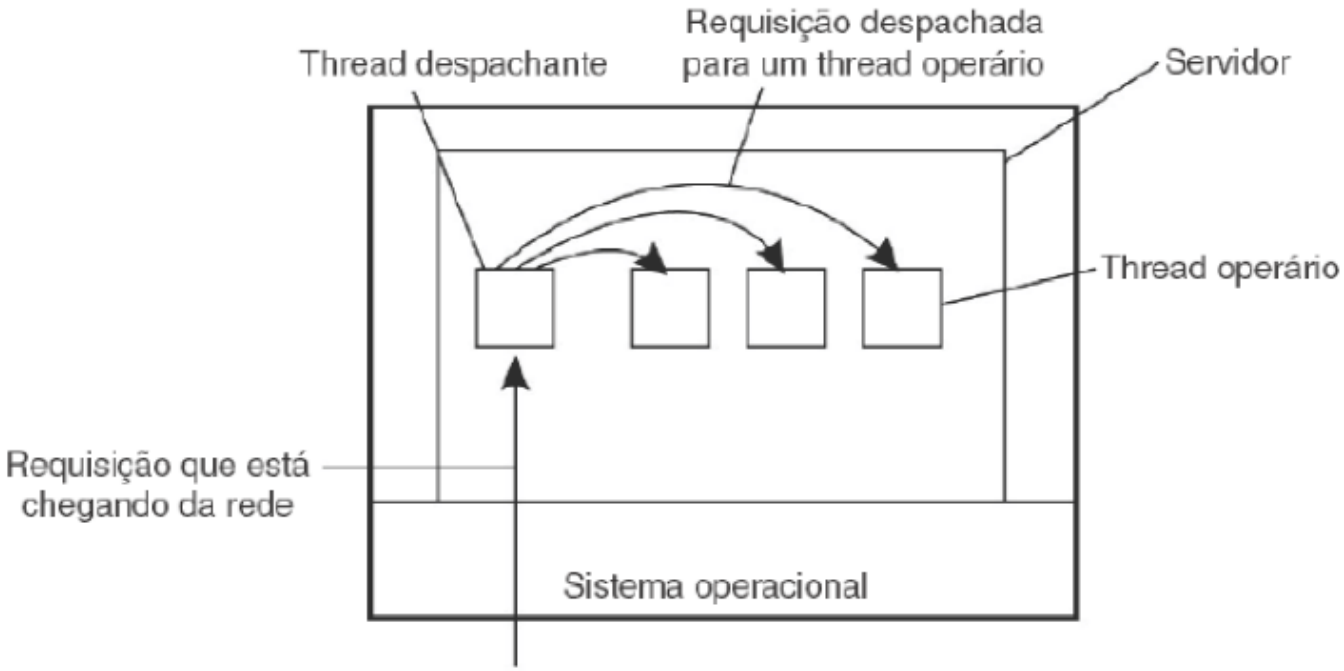
- Browser como **clientes multithread** simplifica, pois:
  - Threads separados são ativados para se encarregar de buscar diferentes partes de uma página;
  - Caso o servidor esteja em sobrecarga, ter um cliente multithread possibilita estabelecer conexões com diferentes servidores, permitindo transmissão dos dados em paralelo;
    - Caso de servidores replicados também aumenta o paralelismo

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- **Servidores Multithreads**
  - Especialmente importante no contexto de computadores multiprocessados
    - Paralelismo ainda mais útil
  - Funcionamento de servidores multithreads:
    - Requisições são enviadas por clientes para uma porta no servidor;
    - **Thread despachante** lê requisições que entram para uma operação de arquivo;
    - Servidor escolhe um **thread operário**;
    - Se o thread escolhido estiver suspenso, outro thread é selecionado para ser executado

PROCESSOS EM SISTEMAS DISTRIBUÍDOS

- **Servidores Multithreads**
  - Funcionamento de servidores multithreads;



  - Threads resultam em considerável ganho de desempenho, mas cada thread é programada sequencialmente, de maneira usual

A API para Protocolos Internet

- **Características da comunicação entre processos**
  - A passagem de mensagens entre um par de processos pode ser suportada por duas operações de comunicação de mensagem:
    - ***send*** e ***receive***;
  - Para que um processo se comunique com outro, um deles envia um (*send*) uma mensagem (uma sequencia de *bytes*) para um destino e o outro processo, no destino, recebe (*receive*) a mensagem.

A API PARA PROTOCOLOS INTERNET

- Características da comunicação entre processos

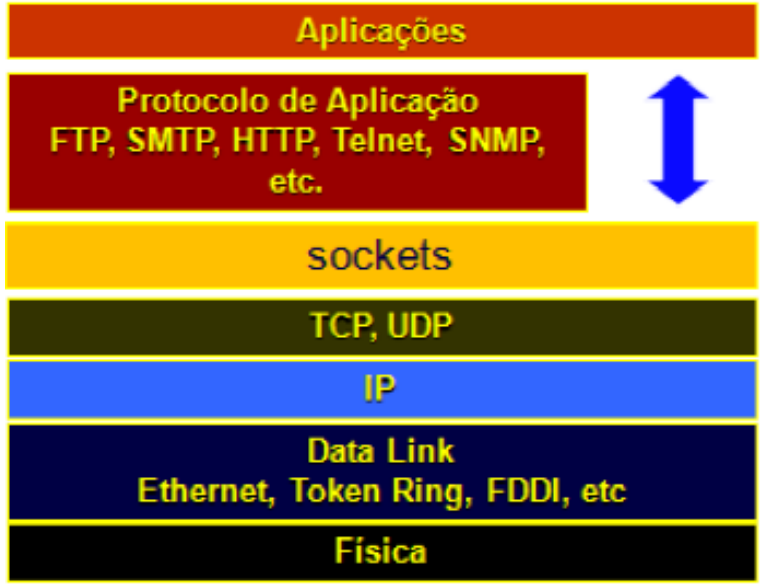
  - **Comunicação Síncrona**
    - Na comunicação síncrona os processos remetente e de destino são sincronizados a cada mensagem;
    - *Send* e *Receive* são operações que causam bloqueio;
    - Quando um envio (*send*) é realizado, o processo remetente (ou *thread*) é bloqueado até que a recepção (*receive*) correspondente seja realizada;
    - Quando uma recepção é executada, o processo (ou *thread*) é bloqueado enquanto a mensagem não chegar.

A API PARA PROTOCOLOS INTERNET

- **Características da comunicação entre processos**
  - **Comunicação Assíncrona**
    - Na comunicação assíncrona o uso da operação ***send*** **é não bloqueante**;
    - O processo remetente pode prosseguir assim que a mensagem tenha sido copiada para o buffer;
    - A operação *receive* pode ter variantes com e sem bloqueio;
      - Não bloqueante: processo fornece um buffer para ser preenchido em *backgroud*
      - Normalmente os sistemas não fornecem a opção não bloqueante
        - Exemplo: Java permite diversas threads em um único processo

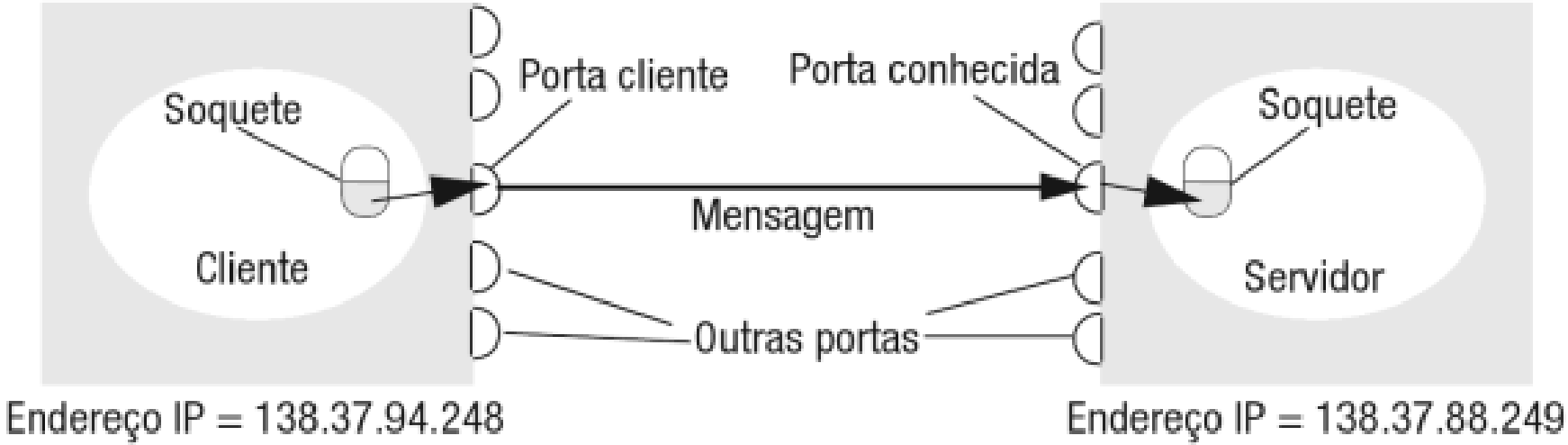
A API PARA PROTOCOLOS INTERNET

- Características da comunicação entre processos
  - **Socket (Soquetes**)
    - As duas formas de comunicação (UDP e TCP) usam a abstração de soquete, um ponto de destino para a comunicação entre processos;
    - A API de Soquetes é uma das mais difundidas para programação sobre a arquitetura TCP/IP.



A API para Protocolos Internet

- Características da comunicação entre processos
  - **Soquetes e Portas**
    - Um processo pode usar o mesmo soquete para enviar e receber;
    - Um processo não pode compartilhar portas com outros processos.



A API para Protocolos Internet

- **Características da comunicação entre processos**
  - **Comunicação por UDP**
    - Um datagrama enviado pelo protocolo UDP é transmitido de um processo origem para um processo destino sem a existência de confirmação ou novas tentativas de envio;
    - Se ocorrer uma falha, a mensagem poderá não chegar;
      - Falha de omissão
    - Para algumas aplicações, é aceitável usar um serviço que esteja exposto a falhas por omissões ocasionais (serviços executados sobre UDP):
      - *Domain Name Service*;
      - Voicer Over IP (VoIP);

A API PARA PROTOCOLOS INTERNET

Cliente UDP

```
import java.net.*;
import java.io.*;
public class UDPClient{
   public static void main(String args[ ]){
      // args fornece o conteúdo da mensagem e o nome de host do servidor
      DatagramSocket aSocket = null;
      try {
            aSocket = new DatagramSocket( );
            byte [ ] m = args[0].getBytes( );
            InetAddress aHost = InetAddress.getByName(args[1]);
            int serverPort = 6789;
            DatagramPacket request =
                  new DatagramPacket(m, m.length( ), aHost, serverPort);
            aSocket.send(request);
            byte[ ] buffer = new byte[1000];
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(reply);
            System.out.println("Reply: " + new String(reply.getData( )));
      } catch (SocketException e){System.out.println("Socket: " + e.getMessage( ));
      } catch (IOException e){System.out.println("IO: "+ e.getMessage( ));
      } finally { if(aSocket!= null) aSocket.close( );}
   }
}
```

A API PARA PROTOCOLOS INTERNET

Servidor UDP

```
import java.net.*;
import java.io.*;
public class UDPServer{
public static void main(String args[ ]){
        DatagramSocket aSocket = null;
        try{
                aSocket = new DatagramSocket(6789);
                byte[ ] buffer = new byte[1000];
               while(true){
                  DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                     aSocket.receive(request);
                     DatagramPacket reply = new DatagramPacket(request.getData( ),
                          request.getLength( ), request.getAddress( ), request.getPort( ));
                  aSocket.send(reply);
                  }
        } catch (SocketException e){System.out.println("Socket: " + e.getMessage( ));
        } catch (IOException e) {System.out.println("IO: " + e.getMessage( ));
        } finally {if (aSocket!= null) aSocket.close( );}
        }
```

A API para Protocolos Internet

- **Características da comunicação entre processos**
  - **Comunicação por TCP**
    - A API do protocolo TCP fornece abstração de um fluxo de bytes no qual dados poder lidos (*receive*) e escritos (*send*);
    - A API TCP pressupõe que, quando dois processos estão estabelecendo uma conexão, um deles desempenha papel de cliente e outro de servidor, mas daí em diante eles poderiam ser iguais;
    - Muitos serviços são executados em conexões TCP com números de porta reservados:
      - *HTTP, FTP, Telnet, SMTP, etc.*

A API PARA PROTOCOLOS INTERNET

Cliente TCP

```
import java.net.*;
import java.io.*;
public class TCPClient {
        public static void main (String args[ ]) {
        // os argumentos fornecem a mensagem e o nome de host de destino
           Socket s = null;
            try{
                int serverPort = 7896;
                s = new Socket(args[1], serverPort);
                DataInputStream in = new DataInputStream(s.getInputStream( ));
                DataOutputStream out =
                        new DataOutputStream( s.getOutputStream( ));
                out.writeUTF(args[0]); // UTF é uma codificação de string; veja a Seção 4.3
                 String data = in readUTF( );
                System.out.println("Received: "+ data);
               }catch (UnknownHostException e){
                System.out.println("Sock:"+e.getMessage( ));
           } catch (EOFException e){System.out.println("EOF:"+e.getMessage( ));
           } catch (IOException e){System.out.println("IO:"+e.getMessage( ));
        } finally {if(s!=null) try {s.close( );}catch (IOException e){/*close falhou*/}}
        }
}
```

A API PARA PROTOCOLOS INTERNET

Servidor TCP

```
import java.net.*;
import java.io.*;
public class TCPServer {
        public static void main (String args[ ]) {
            try{
                int serverPort = 7896;
                ServerSocket listenSocket = new ServerSocket(serverPort);
                while(true) {
                    Socket clientSocket = listen Socket.accept( );
                    Connection c = new Connection(clientSocket);
                }
            } catch(IOException e) {System.out.println("Listen :"+e.getMessage( ));}
        }
 }
```

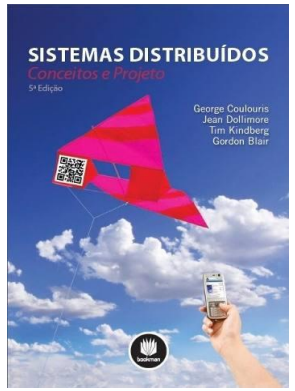
Continua...

A API PARA PROTOCOLOS INTERNET

Servidor TCP

```
class Connection extends Thread {
        DataInputStream in;
        DataOutputStream out;
        Socket clientSocket;
        public Connection (Socket aClientSocket) {
            try {
                clientSocket = aClientSocket;
                in = new DataInputStream(clientSocket.getInputStream( ));
                out =new DataOutputStream(clientSocket.getOutputStream( ));
                this.start( );
            } catch(IOException e) {System.out.println("Connection:"+e.getMessage( ));}
        }
        public void run( ){
            try {                   // an echo server
                String data = in.readUTF( );
                out.writeUTF(data);
            } catch(EOFException e) {System.out.println("EOF:"+e.getMessage( ));
            } catch(IOException e) {System.out.println("IO:"+e.getMessage( ));
            } finally { try {clientSocket.close( );}catch (IOException e){/*close falhou*/ }}
        }
}
```

LIVRO TEXTO

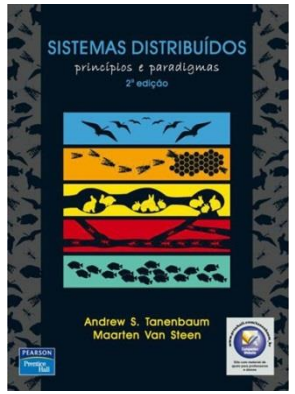


- (Coulouris, 2013)

  COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.

  Sistemas Distribuídos: Conceito e Projeto

  Artmed, 5ª edição, 2013



- (Tanenbaum, 2008)

  TANENBAUM, A. S.; STEEN, M. V.

  Sistemas Distribuídos

  Pearson, 2ª edição, 2008