Sistemas Distribuídos e Programação Concorrente Sockets em Java

Prof. MSc. Rodrigo Malara

Adaptado de Carlos Alberto Kamienski
Djamel Sadok
CIn/UFPE

Identificação de aplicações

- Como cada computador é identificada unicamente na Internet ?
  - Resposta: Endereços IP
- Como a entidade de transporte (TCP) identifica qual aplicação está sendo utilizada na comunicação em rede ?
  - Reposta: Portas TCP/IP

Números de portas

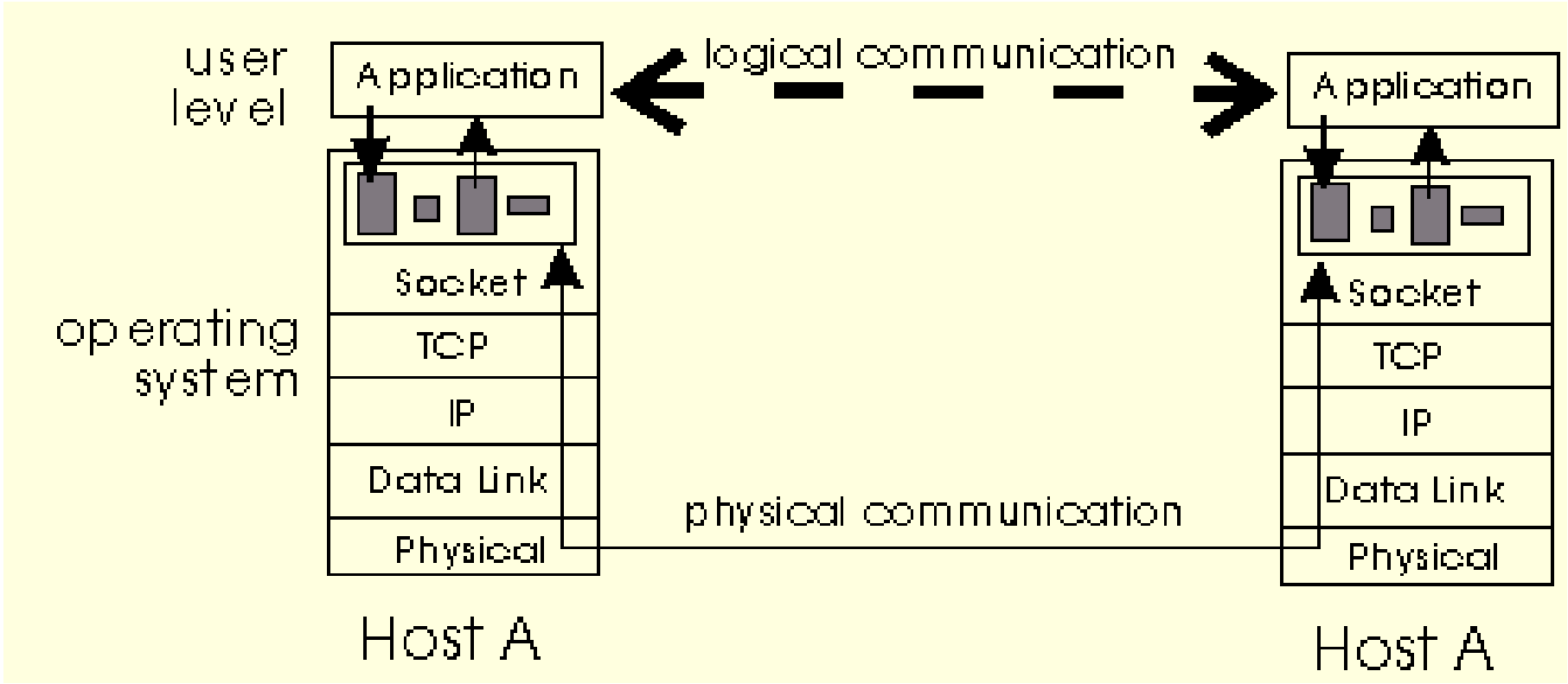
- 1-255 reservadas para serviços padrão portas “bem conhecidas”
- 256-1023 reservado para serviços Unix
- 1-1023 Somente podem ser usadas por usuários privilegiados (super-usuário)
- 1024-9999 Usadas por processos de sistema e de usuário
- 10000- 65536 processos Usadas somente por

de usuário ou atendimento de clientes pelos threads de servidor

Unix BSD Sockets

- Interface padrão para comunicação entre processos em redes TCP/IP
- Nasceu com o Unix de Berkeley
- Os projetistas tentaram usar ao máximo as chamadas de sistema do Unix
- Implementada hoje em vários SOs
- Programar com sockets pode ser visto como **desenvolver um protocolo de aplicação**
- A Comunicação usando Sockets é Full Duplex
  - É possível ler e escrever no socket ao mesmo tempo!

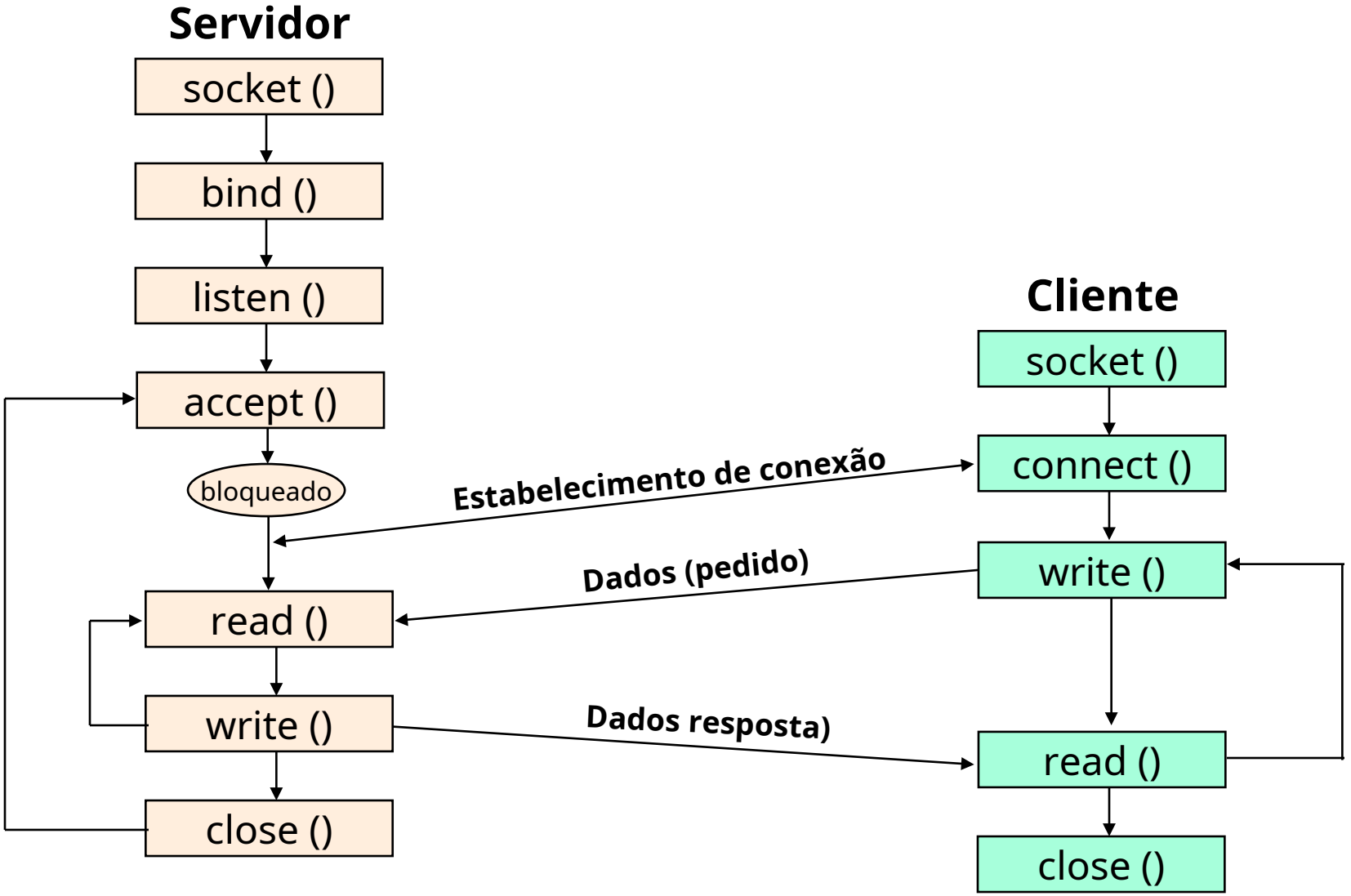
Berkeley Sockets



Tipos de sockets

- Serviço com conexão
  - Implementa um *stream* de dados (SOCK_STREAM)
  - Protocolo TCP (tipicamente)
- Serviço sem conexão
  - Implementa um serviço de datagramas (SOCK_DGRAM)
  - Protocolo UDP (tipicamente)
  - Acessa diretamente a camada de rede (SOCK_RAW)

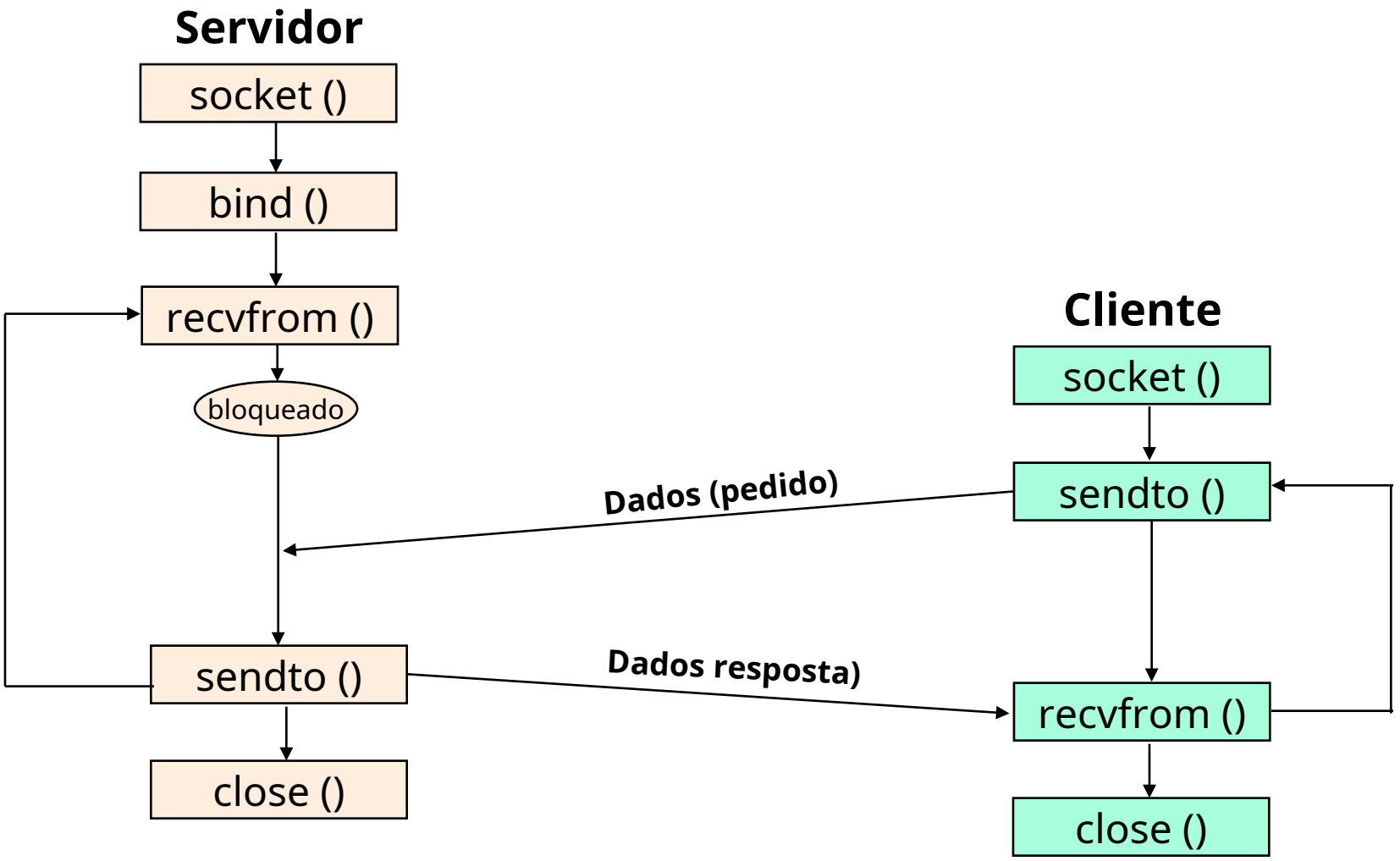
Serviço com Conexão (TCP)



Principais funções da API

socket	Cria um novo descritor para comunicação
connect	Iniciar conexão com servidor
write	Escreve dados em uma conexão
read	Lê dados de uma conexão
close	Fecha a conexão
bind	Atribui um endereço IP e uma porta a um socket
listen	Coloca o socket em modo passivo, para “escutar” portas
accept	Bloqueia o servidor até chegada de requisição de conexão
recvfrom	Recebe um datagrama e guarda o endereço do emissor
sendto	Envia um datagrama especificando o endereço

Serviço sem Conexão (UDP)



Sockets em Java

- Java modernizou a API para trabalhar com sockets
- O programador não precisa chamar todas as funções, algumas chamadas são automáticas
- Exemplos
  - Socket: equivalente a *socket* e *bind*
  - ServerSocket: equivalente a *socket*, *bind* e *listen*
- Sockets são implementados no pacote ***java.net***
- A transmissão e o envio de dados são feitos através de classes do pacote ***java.io*** de maneira semelhante à escrita e leitura em arquivos
  - Classes *DataInputStream*, *DataOutputStream*, etc.,

```
import java.net.*;
import java.io.*;

public class SimpleJavaClient {
   public static void main(String[] args)  {
          try {
                Socket s = new Socket("127.0.0.1", 9999);
                InputStream i = s.getInputStream();
                OutputStream o = s.getOutputStream();
                String str;
                do {
                      byte[] line = new byte[100];
                      System.in.read(line);
                      o.write(line);
                      i.read(line);
                      str = new String(line);
                      System.out.println(str.trim());
                } while ( !str.trim().equals("bye") );
                s.close();
          }
          catch (Exception err) {
                System.err.println(err);
          }
   }
}
```

Conectando com o servidor

```
import java.net.*;
import java.io.*;

public class SimpleJavaClient {
   public static void main(String[] args)  {
          try {
                Socket s = new Socket("127.0.0.1", 9999);
                InputStream i = s.getInputStream();
                OutputStream o = s.getOutputStream();
                String str;
                do {
                      byte[] line = new byte[100];
                      System.in.read(line);
                      o.write(line);
                      i.read(line);
                      str = new String(line);
                      System.out.println(str.trim());
                } while ( !str.trim().equals("bye") );
                s.close();
          }
          catch (Exception err) {
                System.err.println(err);
          }
   }
}
```

Pegando canais de envio e recepção de dados

```
import java.net.*;
import java.io.*;

public class SimpleJavaClient {
   public static void main(String[] args)  {
          try {
                 Socket s = new Socket("127.0.0.1", 9999);
                 InputStream i = s.getInputStream();
                 OutputStream o = s.getOutputStream();
                 String str;
                 do {
                        byte[] line = new byte[100];
                        System.in.read(line);
                        o.write(line);
                        i.read(line);
                        str = new String(line);
                        System.out.println(str.trim());
                 } while ( !str.trim().equals("bye") );
                 s.close();
          }
          catch (Exception err) {
                 System.err.println(err);
          }
   }
}
```

Usuário digita algo

```
import java.net.*;
import java.io.*;

public class SimpleJavaClient {
   public static void main(String[] args)  {
          try {
                Socket s = new Socket("127.0.0.1", 9999);
                InputStream i = s.getInputStream();
                OutputStream o = s.getOutputStream();
                String str;
                do {
                      byte[] line = new byte[100];
                      System.in.read(line);
                      o.write(line);
                      i.read(line);
                      str = new String(line);
                      System.out.println(str.trim());
                } while ( !str.trim().equals("bye") );
                s.close();

          atch (Exception err) {
                System.err.println(err);
          }
   }
}
```

Envia o que foi digitado e recebe resposta

```
import java.net.*;
import java.io.*;

public class SimpleJavaClient {
   public static void main(String[] args)  {
          try {
                Socket s = new Socket("127.0.0.1", 9999);
                InputStream i = s.getInputStream();
                OutputStream o = s.getOutputStream();
                String str;
                do {
                      byte[] line = new byte[100];
                      System.in.read(line);
                      o.write(line);
                      i.read(line);
                      str = new String(line);
                      System.out.println(str.trim());
                } while ( !str.trim().equals("bye") );
                s.close();
          }
          catch (Exception err) {
                System.err.println(err);
          }
   }
}
```

Exibe o que foi recebido

```
import java.net.*;
import java.io.*;

public class SimpleJavaClient {
   public static void main(String[] args)  {
          try {
                 Socket s = new Socket("127.0.0.1", 9999);
                 InputStream i = s.getInputStream();
                 OutputStream o = s.getOutputStream();
                 String str;
                 do {
                        byte[] line = new byte[100];
                        System.in.read(line);
                        o.write(line);
                        i.read(line);
                        str = new String(line);
                        System.out.println(str.trim());
                 } while ( !str.trim().equals("bye") );
                 s.close();
          }
          catch (Exception err) {
                 System.err.println(err);
          }
   }
}
```

Fecha a conexão

```
import java.io.*;
import java.net.*;

public class SimpleJavaServer {
   public static void main(String[] args)  {
          try {
                 ServerSocket s = new ServerSocket(9999);
                 String str;
                 while (true) {
                        Socket c = s.accept();
                        InputStream i = c.getInputStream();
                        OutputStream o = c.getOutputStream();
                        do {
                               byte[] line = new byte[100];
                               i.read(line);
                               o.write(line);
                               str = new String(line);
                        } while ( !str.trim().equals("bye") );
                        c.close();
                 }
          }
          catch (Exception err){
             System.err.println(err);
          }
   }
}
```

Exercício

- Utilizando o código do slide anterior, criar um socket que conecte no www.google.com.br na porta 80
- Envie o comando GET para o servidor
- Exiba no console a resposta obtida do servidor

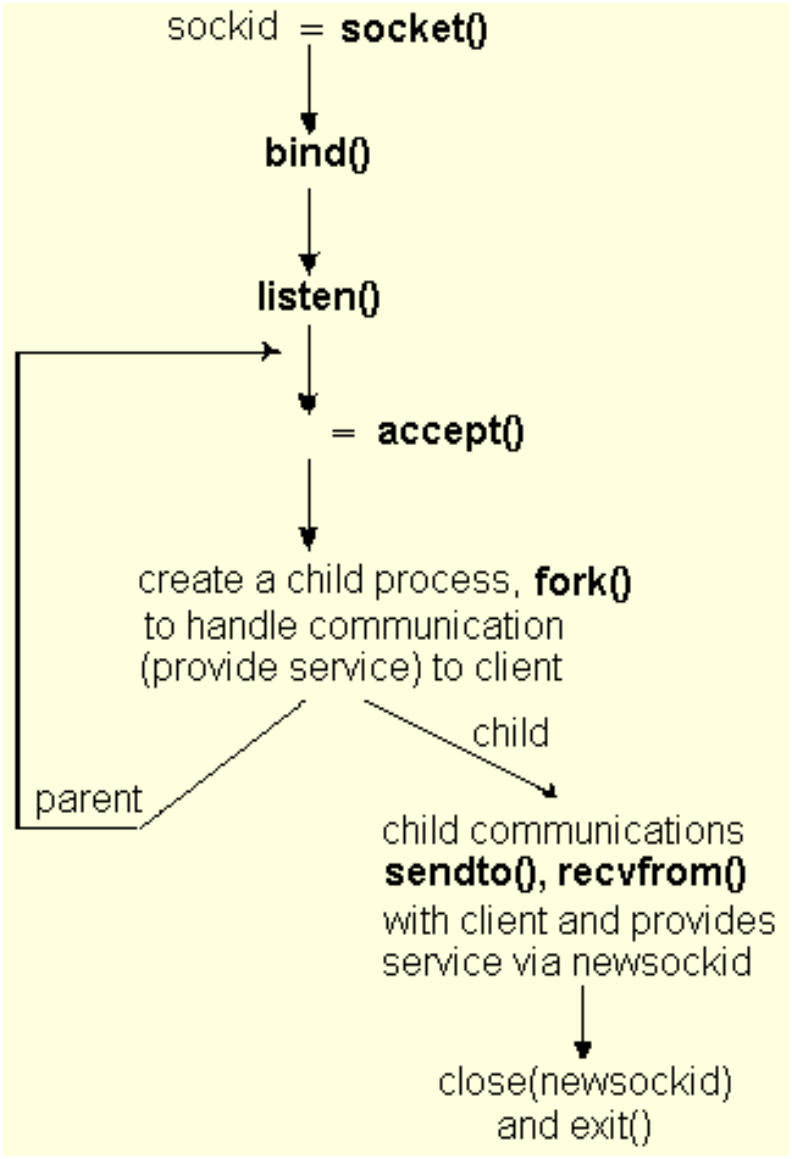
Identificação de aplicações

- Um cliente pode abrir várias conexões com o mesmo servidor (ex. páginas web) concorrentemente.
- Como o cliente sabe para qual programa enviar os pacotes?
  - Resposta: A cada conexão, um worker thread é atribuído no servidor para fazer o atendimento da conexão do cliente e a comunicação ocorre em uma porta superior que é atribuída ao estabelecer a conexão.

Servidor: lidando com vários clientes

- O programa servidor criado anteriormente não lida com vários clientes. Apenas um de cada vez.
- É necessário utilizar Threads para se lidar com vários clientes ao mesmo tempo
  - A comunicação com cada cliente deve ser delegada a um thread

Estrutura Típica de um Servidor



```
import java.io.*;
import java.net.*;

public class SimpleJavaServer {
   public static void main(String[] args)  {
          try {
                ServerSocket s = new ServerSocket(9999);
                String str;
                while (true) {
                      Socket c = s.accept();
                      InputStream i = c.getInputStream();
                      OutputStream o = c.getOutputStream();
                      do {
                            byte[] line = new byte[100];
                            i.read(line);
                            o.write(line);
                            str = new String(line);
                      } while ( !str.trim().equals("bye") );
                      c.close();
                }
          }
          catch (Exception err){
             System.err.println(err);
          }
   }
}
```

Código de comunicação com o cliente. Deve ser executado em outro thread

Exercício

- Utilizando o código disponibilizado como exemplo, criar um servidor que receba conexões na porta 4444.
- Ao receber uma conexão, ele verifica se o cliente enviou a string “Oi”.
  - Se sim, responda com “Olá, como posso ajudar ?”
  - Se não, responda com “O que você precisa ?”
  - Se for digitada a palavra “tchau”, finalizar a conexão
- Crie um novo programa cliente que conecte no servidor e teste as duas situações
- Exiba no console a resposta obtida do servidor

Trabalho

- Transforme o programa servidor anterior em multithread.
- Dicas:
  - Crie a classe chamada Worker
    - Ela deve implementar a interface Runnable

```
public class Worker implements Runnable {
```

    - Ter um construtor que aceite um Socket
      - Armazenar o socket recebido em um atributo privado

```
public Worker(Socket socket) {
    this.socket = socket;
}
```

  - Copiar trecho marcado slide anterior para método run()
    - Acertar detalhes

```
public void run() {
    // codigo aqui
}
```

Trabalho

- Transforme o programa servidor anterior em multithread.
- Dicas:
  - Modificar loop da classe SimpleJavaServer para:
    - remover o código copiado para a classe Worker
    - criar um objeto da classe Worker

      ```
      Worker worker = new Worker(c);
      ```
    - crie um thread passando o objeto criado no constructor

      ```
      Thread t = new Thread(worker);
      ```
    - Inicie o thread

      ```
      t.start();
      ```

Sockets sem Conexão (Java)

- Cliente:
  - `socket = new DatagramSocket( );`
  - `message = new DatagramPacket(msg,length,Addr,Port);`
  - `reply = new DatagramPacket( new byte[100], 100 );`
  - `socket.send( message );`
  - `socket.receive( reply );`
  - `socket.close();`
- Servidor:
  - `socket = new DatagramSocket(porta);`
  - `socket.receive( message );`
  - `socket.send( message );`

Comunicação em Grupo
IP multicast Java API

- MulticastSocket (subclasse de DatagramSocket).
- Adiciona a capacidade de se entrar e sair de grupos multicast.

Group communication – IP multicast peers (1)

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
  public static void main(String args[]){
    // args possui a mensagem e o IP do
    //grupo multicast (ex. "228.5.6.7")
    try {
    InetAddress group = InetAddress.getByName("228.5.6.7");
    MulticastSocket s = new MulticastSocket(6789);
    s.joinGroup(group);
    byte [] m = “Teste”.getBytes();
    DatagramPacket messageOut =
      new DatagramPacket(m, m.length, group, 6789);
    s.send(messageOut);

    //Continua no próximo slide
```

Group communication – IP multicast peers (2)

```
// pega bytes de outros do grupo
byte[] buffer = new byte[1000];
for(int i=0; i< 3; i++) {
  DatagramPacket messageIn =
    new DatagramPacket(buffer, buffer.length);
  s.receive(messageIn);
  System.out.println("Received:" +
    new String(messageIn.getData()));
}
s.leaveGroup(group);
s.close();
}catch (SocketException e){
  System.out.println("Socket: " + e.getMessage());
}catch (IOException e){
  System.out.println("IO: " + e.getMessage());}
}
}
```