Programação com "sockets" (em Java)

Prof. Dr. Julio Arakaki

Depto. Ciência da Computação

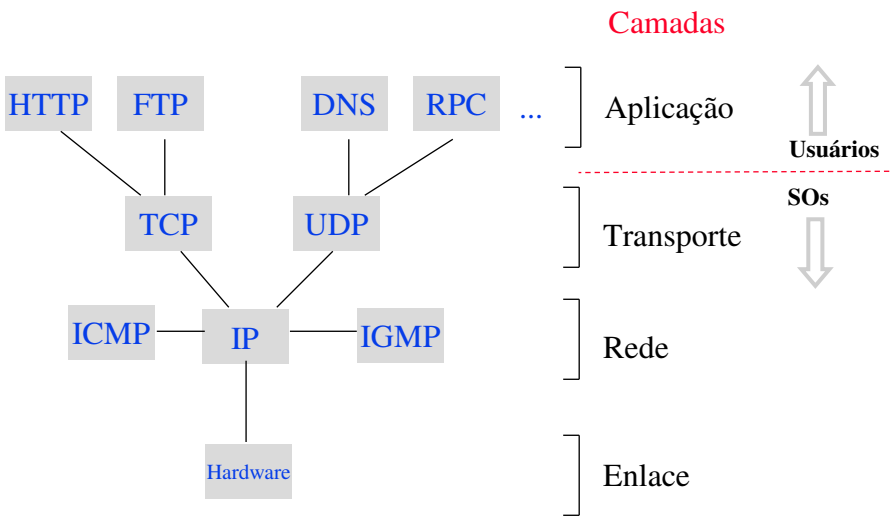
Comunicação entre processos / aplicações

- "Sockets" é um dos mecanismos mais utilizados

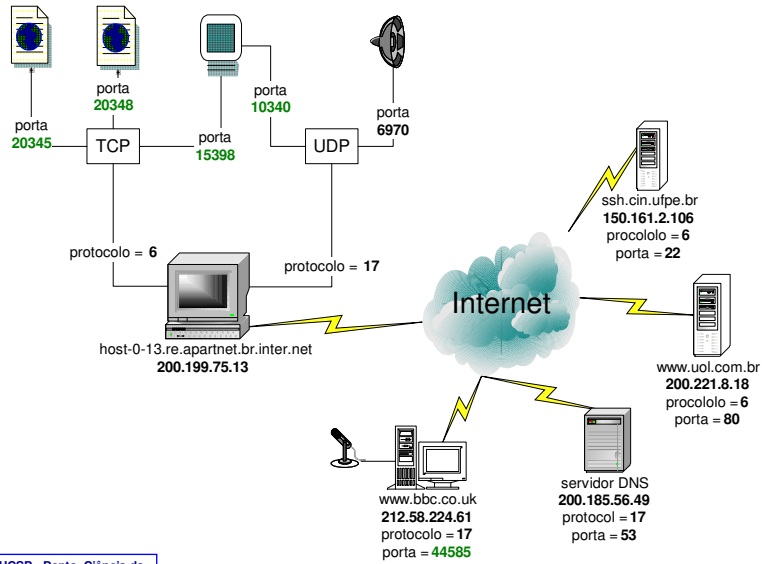
- Java oferece modos de utilização:
  . TCP ("Transmission Control Protocol") – orientado a conexão
  . UDP ("User Datagram Protocol") – orientado a datagrama
  . Entre outros

  Funcionam sobre o protocolo IP ("Internet Protocol")

Principais protocolos (Rede/Internet)



Identificação na "rede"



Programação (em Rede/Internet)

- Sockets
  - envia/recebe (*send/receive*)
  - Característica: eficiência
- RPC ("Remote Procedure Call")
  - Chamada remota de procedimento
  - Característica: transparência e facilidade de programação
- Objetos distribuídos
  - Características: transparência, facilidade e todos os benefícios da programação orientada a objetos, execução mais lenta
  - Exemplos: DCOM, CORBA, Java RMI, EJB, Servlets, Frameworks (Struts, Spring, ...), Web Services, etc.

Sockets – BSD Unix

- Interface padrão para comunicação entre processos em redes TCP/IP
- Implementação no SO: Unix de Berkeley
- A maioria dos SOs implementam os sockets
- Programar com sockets pode ser visto como:

  Desenvolver um protocolo de Comunicação

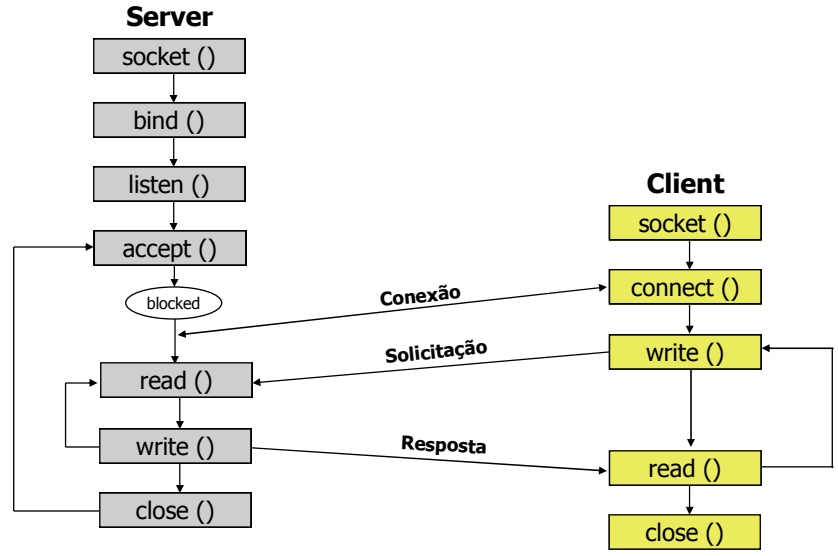
Principais tipos de sockets

- 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 de baixo nível
  - Protocolo IP (tipicamente)

Principais funções da API de Sockets

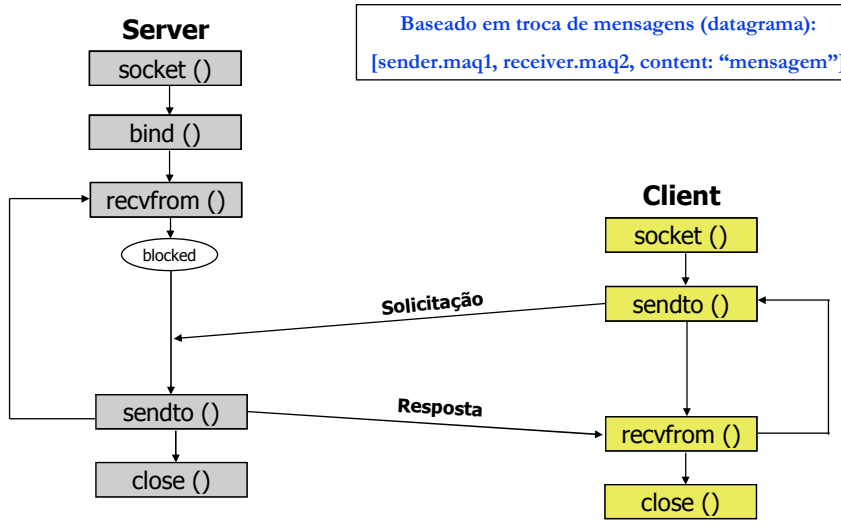
socket	Cria um novo descritor para comunicação
connect	Iniciar conexão com servidor
write	Escreve dados em uma conexão
read	Lê dados de uma conexão
close	Fecha a conexão
bind	Atribui um endereço IP e uma porta a um socket
listen	Coloca o socket em modo passivo, para "escutar" portas
accept	Bloqueia o 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 com conexão (TCP)

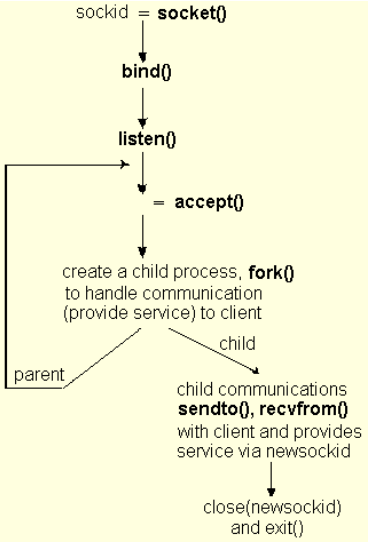


Serviço sem conexão (UDP)

Baseado em troca de mensagens (datagrama):
[sender.maq1, receiver.maq2, content: "mensagem"]



Servidor (estrutura típica)



Codificação de portas (valores/significado/utilização)

- **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 - 4999** — **Usadas por processos de sistema e de usuário**
- **5000 -** — **Usadas somente por processos de usuário**

Sockets em Java

- Em Java, algumas chamadas (de funções) são automáticas
- Exemplos:
  - **Socket**: equivalente a ***socket*** e ***bind***
  - **ServerSocket**: equivalente a ***socket***, ***bind*** e ***listen***
- Sockets são implementados no pacote *java.net*
- O envio e a recepção dos dados são realizados através de classes do pacote *java.io* de maneira semelhante à escrita e leitura em arquivos
  - ***DataInputStream***, ***DataOutputStream***, etc.

Servidor TCP

```
            ...
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);
            ...
```

Cliente TCP

```
                …
        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);
                    …
```


Servidor UDP

```
import java.net.*;
import java.util.*;

class ServidorUDP {

  public static void main( String args[] ) throws Exception {

    DatagramSocket socket = new DatagramSocket(7);
    DatagramPacket packet = new DatagramPacket(new byte[512],512);

    while ( true ) {
      socket.receive( packet );
      System.out.println( ""+new Date()+""+packet.getAddress()+
                ":"+packet.getPort()+" mensagem = "+
                new String(packet.getData(),0,packet.getLength()) );
      socket.send( packet );
    }

  }
}
```

Cliente UDP

```
import java.net.*;
import java.util.*;

class ClienteUDP {

  public static void main( String args[] ) throws Exception {
    DatagramSocket socket = new DatagramSocket();
    socket.setSoTimeout( 5000 );
    String mens = "12314315145125231523535";
    byte[] buffer = mens.getBytes();
    DatagramPacket packet = new DatagramPacket(
       buffer,buffer.length,InetAddress.getByName("localhost"),7);
    socket.send( packet );
    Date timeSent = new Date();
    socket.receive( packet );
    Date timeReceived = new Date();
    System.out.println( ""+
        (timeReceived.getTime()-timeSent.getTime())+" ms "+
               new String(packet.getData(),0,packet.getLength()) );
  }
}
```