Interface de Programação *Sockets*

Hermes Senger

Resumo

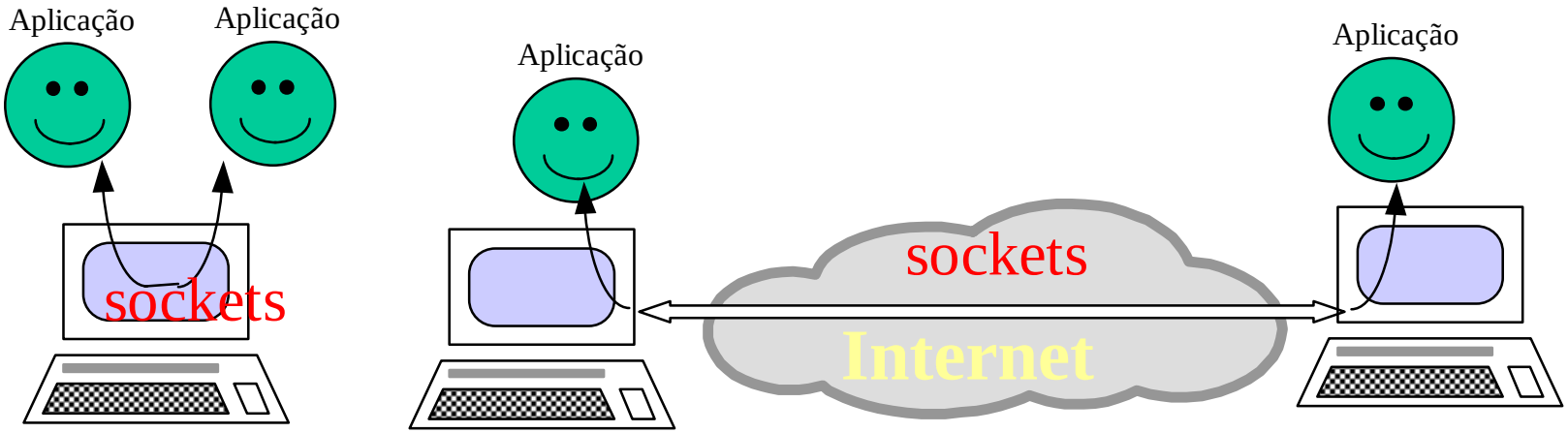
- Motivação
- Introdução
- Histórico
- Protocolos
- Conexões
- O modelo cliente/Servidor
- Exemplo de um servidor
- Exemplo de um cliente
- Chamadas da API *socket*
- Sockets em Java
- Exemplos em Java : cliente e servidor
- Especificação de um projeto

Motivação

- Por que aprender a programar com *sockets* ?
  - ☞Programar na Internet
  - ☞Programar em rede local
  - ☞A interface *socket* tornou-se quase tão popular entre os programadores quanto a própria Internet.
  - ☞Cada vez mais aplicações estão utilizando
  - ☞Isso deve ser muito importante (pelo menos) nos próximos dez anos.

Introdução

- Como é que as aplicações se comunicam na Internet ?
- O que significa Programação para a Internet ?
- O que significa programação com TCP/IP ?



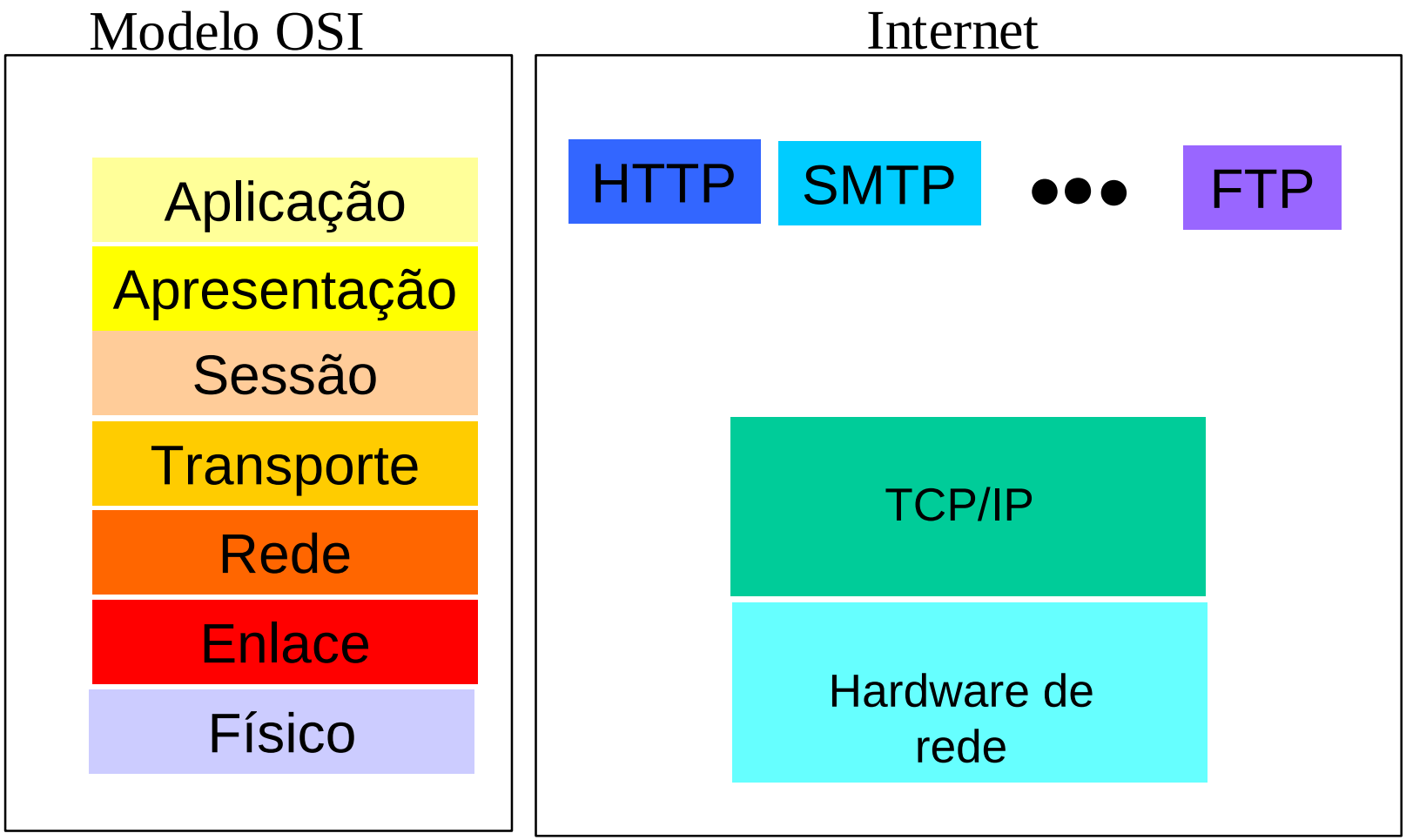
Histórico

- Surgiu pela primeira vez no UNIX (4.1cBSD)
- Posteriormente foi aprimorado no 4.2BSD
- Foi adotado como interface de programação padrão para Internet
- Posteriormente outros S.O. adotaram :
  - Windows - (*Winsock*)
- Permitem escrever programas independente da plataforma
- Outros protocolos : IPX, NetBIOS, ...

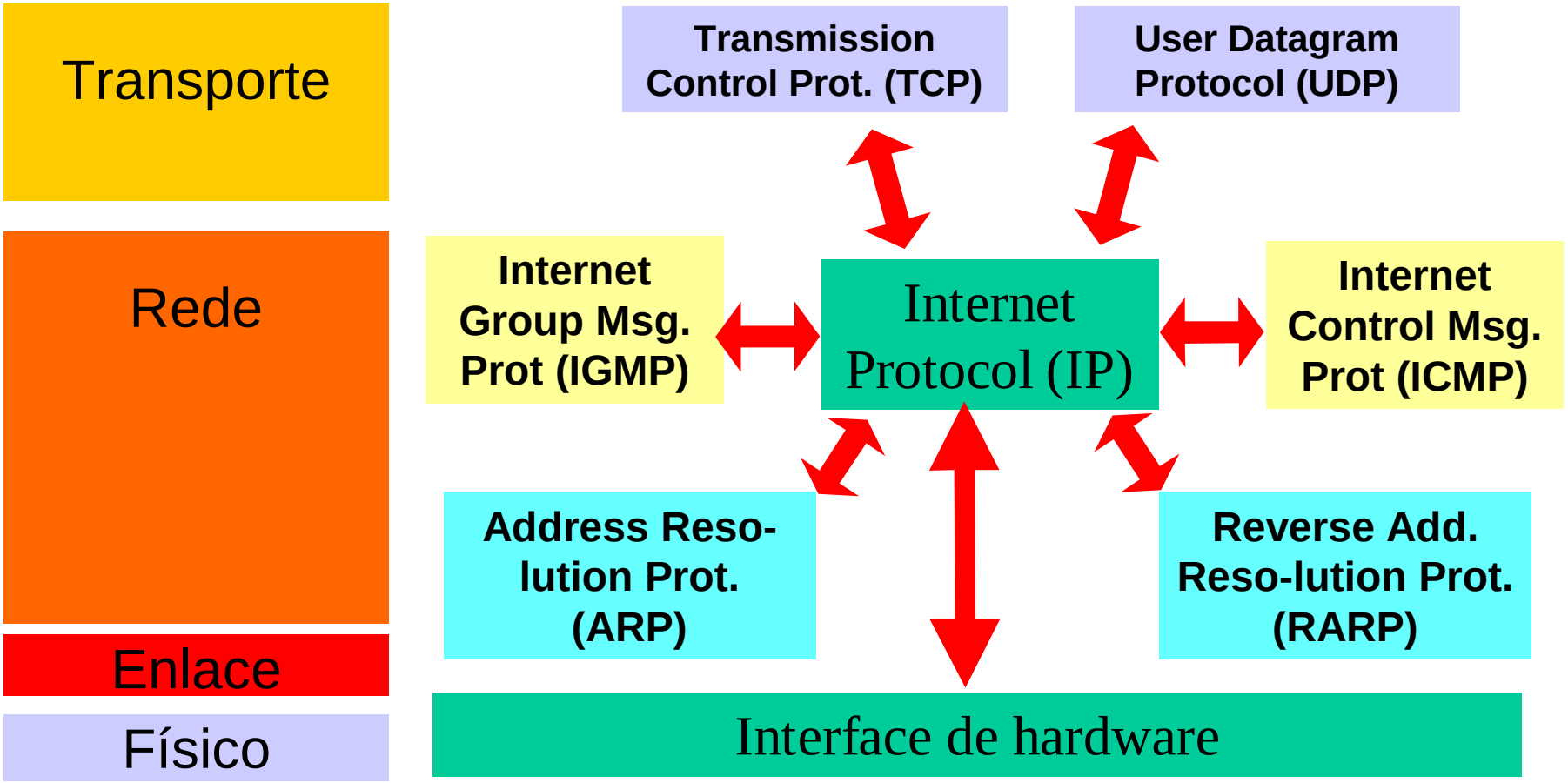
O que são protocolos de comunicação ?

- São regras para a comunicação entre duas aplicações, computadores, dispositivos, etc.
- Em programação, protocolos são funções que implementam serviços de comunicação
- São a interface de um módulo ou de uma camada de rede
- Exemplo : TCP/IP, Telnet, HTTP, SMTP, FTP, NNTP, SNMP, ARP, RARP, BOOTP, ICMP, IGMP,

Protocolos


Modelo OSI
Aplicação
Apresentação
Sessão
Transporte
Rede
Enlace
Físico
Internet
HTTP
SMTP
●●●
FTP
TCP/IP
Hardware de rede

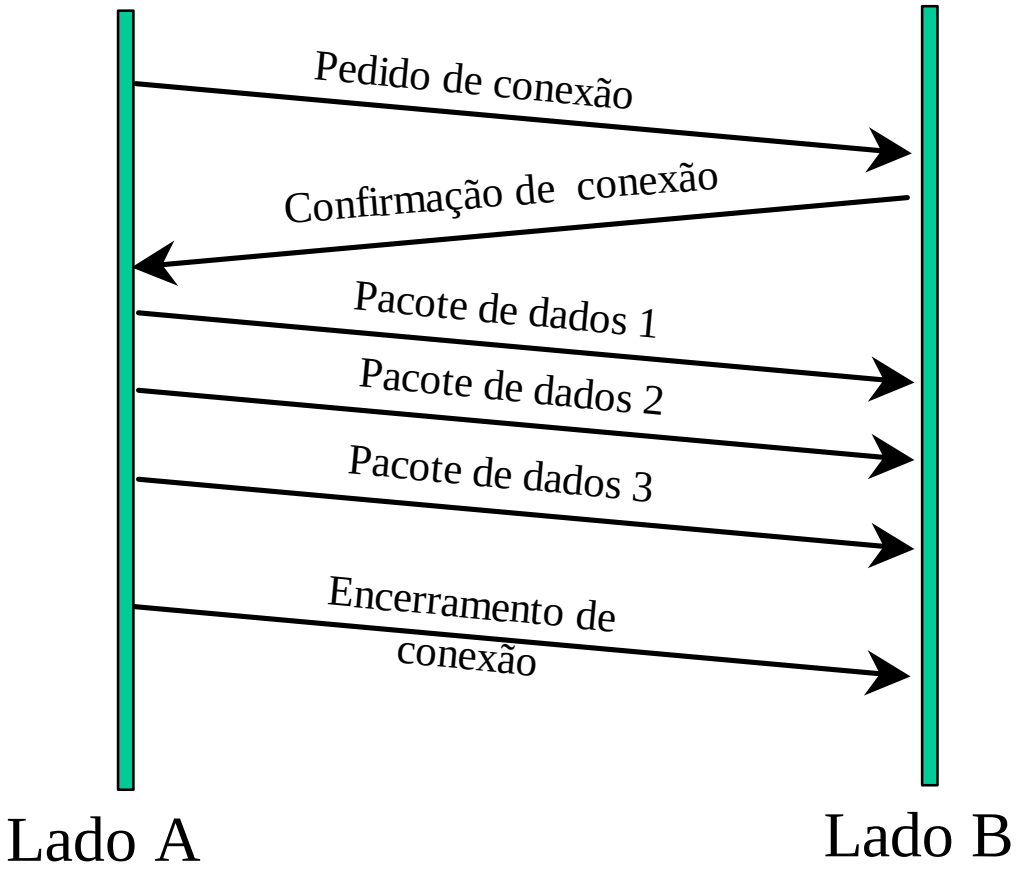
A família TCP/IP



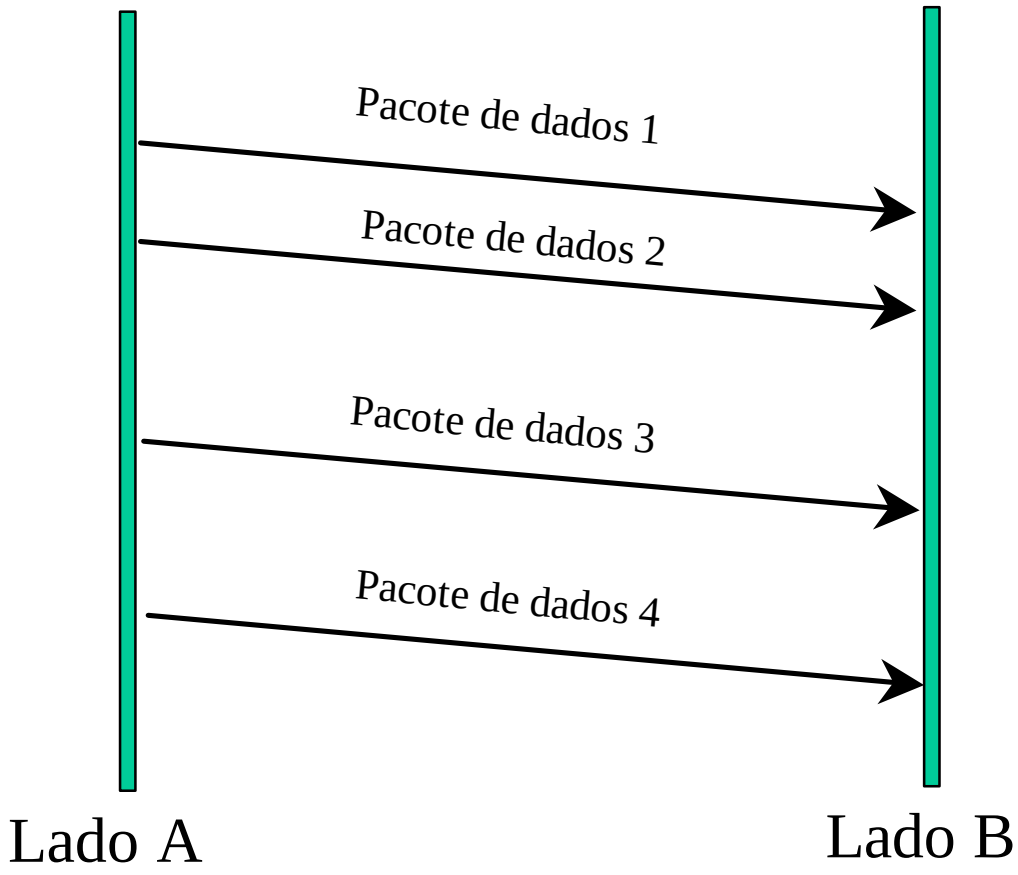
Conexões

- Conceitualmente, há dois tipos de comunicação
  - sem conexão : cliente e servidor podem trabalhar sem esperar pelo outro. Exemplos desse tipo de comunicação: correio comum, correio eletrônico, ...
  - com conexão : ambos os lados precisam abrir uma conexão para depois poderem enviar e receber dados. Exemplo : ligação telefônica, *chat*,...
- Os dois tipos estão disponíveis quando utilizamos *sockets* :
  - com conexão : usa o protocolo TCP
  - sem conexão : usa o protocolo UDP

Comunicação com conexão



Comunicação sem conexão



Conceitos importantes

- Endereço IP : a Internet utiliza endereços IP para identificar univocamente cada computador na rede. Por exemplo, o endereço de ***www.usp.br*** é 143.107.253.62
- Porta: É um endereço para conexão em um particular computador. Um mesmo computador pode receber conexões para diversas aplicações. Normalmente a porta identifica com qual aplicação estamos falando. Várias delas são reservadas, por exemplo, 23 é a porta de Telnet, 25 é SMTP, 80 do HTTP, etc. Portas de 1024 a 65535 são livres. Uma porta é identificada por um inteiro sem sinal de 16 bits (*unsigned int*)

O modelo Cliente/Servidor

- *Sockets* se baseia no modelo cliente/servidor
  - O cliente é quem toma a iniciativa de abrir conexão
  - O servidor fica aguardando pedidos de conexão
- É como quando uma pessoa faz uma chamada telefônica para outra.
- O cliente precisa saber da existência do servidor, e além disso, **conhecer seu endereço**
- O servidor não precisa saber da existência do cliente previamente.
- Por isso, os procedimentos são um pouco diferentes nos dois lados.

Tipos de *socket*

- *Stream socket* :
  - trata a comunicação como u **fluxo contínuo** de bytes entre cliente e servidor.
  - Utiliza o protocolo **TCP** : garante a chegada de tudo que for enviado, e na ordem correta
- *Datagram sockets* :
  - baseado em fluxo de mensagens (datagramas): cada mensagem é enviada isoladamente
  - Utiliza o protocolo **UDP** : que não garante a entrega no destino, e não garante que as mensagens chegem na mesma ordem do envio

O servidor de *sockets*

- Cria um *socket* com a chamada ***socket( )***
- Associa (em inglês "*bind*") o *socket* a um endereço (IP+número de porta) : ***bind( )***
- Prepara-se para receber conexões ***listen( )***
- Espera por um cliente que solicite abertura de conexão :
  - para isto utiliza a chamada e em seguida ***accept( )***
  - essa chamada mantém o servidor bloqueado até a chegada de uma solicitação de conexão
- recebe dados do cliente:***read( )*** ou ***recv( )***
- envia dados para o cliente:***write( )*** ou ***send( )***

Exemplo de um servidor

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
main()
{  int cont,cria_socket,novo_socket,tamanho;
   int tam_buffer = 1024;
   char *buffer = malloc(tam_buffer);
   struct sockaddr_in endereco;
   printf("\x1B[2J");
   if ((cria_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
     printf("Um socket foi criado\n");

   endereco.sin_family = AF_INET;
   endereco.sin_addr.s_addr = INADDR_ANY;
   endereco.sin_port = htons(15000);
```

Exemplo de um servidor

Prepara para receber conexões

```
if (bind(cria_socket,(struct sockaddr *)&endereco,sizeof(endereco))==0)
  printf("Bind OK\n");

listen(cria_socket,3);
tamanho = sizeof(struct sockaddr_in);

novo_socket = accept(cria_socket,(struct sockaddr*)&endereco,&tamanho);
if(novo_socket > 0)printf(”Cliente %s conectado\n",inet_ntoa
(endereco.sin_addr));

recv(novo_socket,buffer,tam_buffer,0);
printf(”Mensagem enviada pelo cliente %s\n",buffer);

strcpy(buffer,"Sua mensagem foi recebida com sucesso");
send(novo_socket,buffer,tam_buffer,0);
close(novo_socket);
close(cria_socket);
}
```

Prepara para receber conexões

Aguarda nova conexão

Lê dados

Envia dados

O cliente

- Cria um *socket* com a chamada ***socket ( )***
- Conecta o *socket* ao servidor ***connect( )***
- Envia/recebe dados:
  - há várias maneiras de fazer isso
  - a mais comum é usar as chamadas ***read( )*** e ***write( )***
  - mas ***send( )*** e ***recv( )*** também são utilizados
- Encerrar a conexão : ***close( )***

Um exemplo de cliente

```
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
main(int argc,char *argv[])
{
   int cria_socket;
   int tam_buffer = 1024;
   char *buffer = malloc(tam_buffer);
   struct sockaddr_in endereco;

   printf("\x1B[2J");
   if ((cria_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
     printf("The Socket was created\n");
   endereco.sin_family = AF_INET;
   endereco.sin_port = htons(15000);
   inet_pton(AF_INET,argv[1],&endereco.sin_addr);
```

Exemplo de cliente (cont.)

```
if(connect(cria_socket,(struct sockaddr *)&endereco,sizeof(endereco))==0)
  printf(”Conectado c/o servidor %s...\n",inet_ntoa (endereco.sin_addr));
```

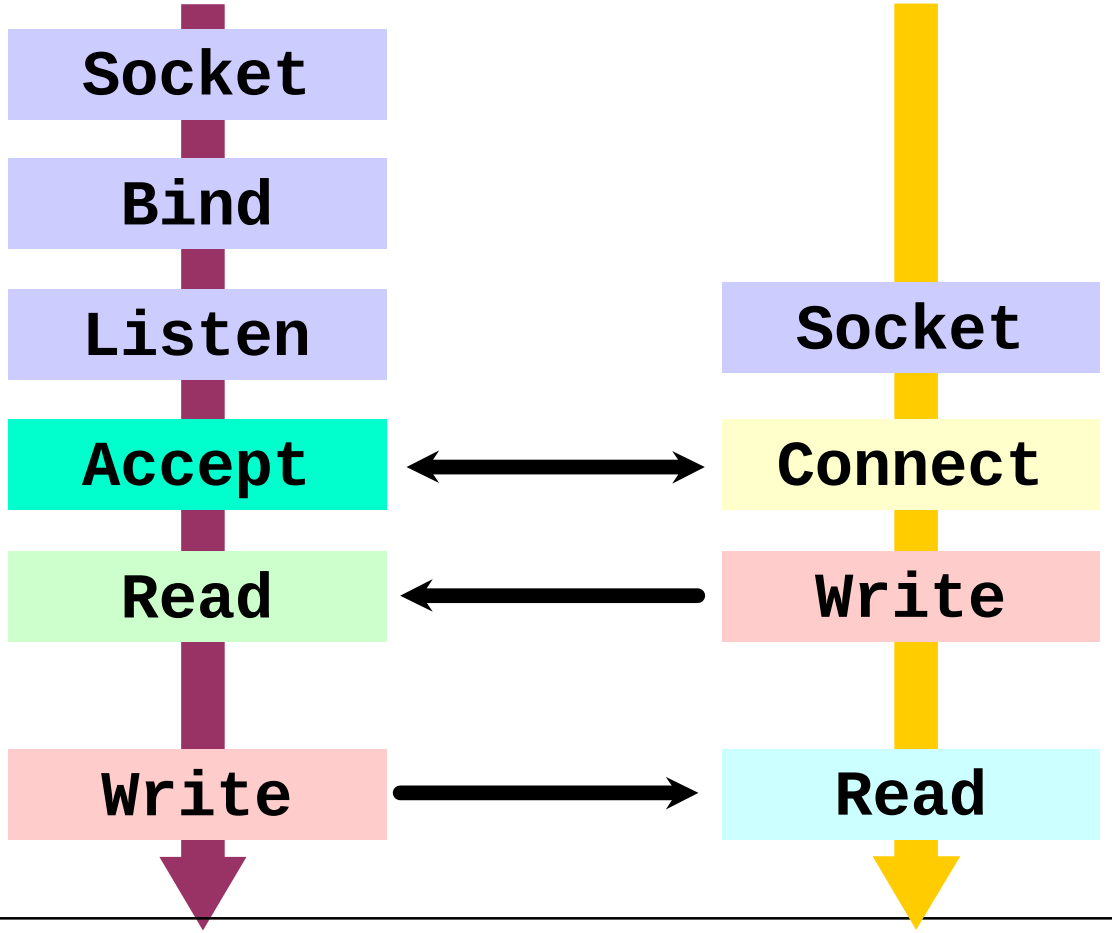

```
printf("Forneça a mensagem a ser enviada: ");
gets(buffer);
send(cria_socket,buffer,tam_buffer,0);
```


```
recv(cria_socket,buffer,tam_buffer,0);
```


```
printf("Resposta recebida do servidor: %s\n",buffer);

close(cria_socket);
}
```

Servidor x Cliente



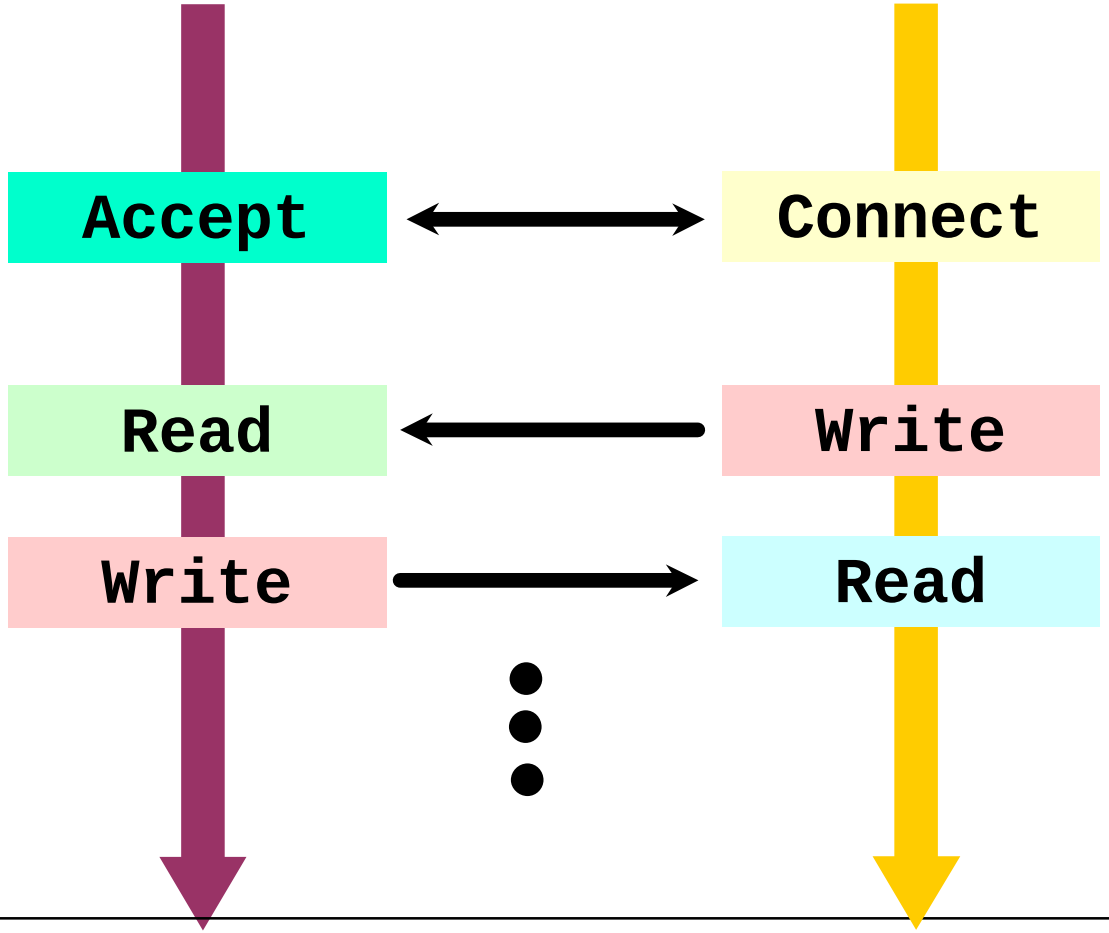
Existem outras chamadas à API

- ***read( )*** :utilizada para ler mensagens de um socket
- ***write( )***: utilizada para enviar dados através de uma conexão socket
- ***sendto( )*** e ***sendmsg( )*** : enviar dados em qualquer tipo de comunicação, inclusive não-orientada a conexão - neste caso requer um parâmetro com o endereço de quem irá receber a mensagem.
- ***recvfrom( ) e recvmsg( )***: receber dados em qualquer tipo de comunicação, inclusive não-orientada a conexão - neste caso requer um parâmetro com o endereço do transmissor.
- Sockets tambem podem ser não-bloqueantes

Comunicação sem conexão

- Nesta modalidade ninguém abre conexão
- Ambos os lados criam um socket : ***socket( )***
- Associam-no a uma porta : ***bind( )***
- E iniciam a fase de envio e recebimentos : com ***sendto( )*** e ***recvfrom( )***, por exemplo

Servidor x Cliente



Existem outras chamadas à API

- ***read( )*** :utilizada para ler mensagens de um socket
- ***write( )***: utilizada para enviar dados através de uma conexão socket
- ***sendto( )*** e ***sendmsg( )*** : enviar dados em qualquer tipo de comunicação, inclusive não-orientada a conexão - neste caso requer um parâmetro com o endereço de quem irá receber a mensagem.
- ***recvfrom( ) e recvmsg( )***: receber dados em qualquer tipo de comunicação, inclusive não-orientada a conexão - neste caso requer um parâmetro com o endereço do transmissor.
- Sockets tambem podem ser não-bloqueantes

Sockets em JAVA

- A linguagem JAVA possui recursos próprios para utilização de *sockets*: o pacote ***java.net.*** Tais recursos incluem:
  - protocolos de aplicação **FTP, HTTP e TELNET**
  - protocolos de transporte **TCP** e **UDP**
  - uma classe para manipulação de URL's

Clases do pacote ***java.net***

- ***ServerSocket*** :define um servidor de *sockets*. Toda aplicação que deseje implementar um servidor de *sockets* deve instanciar esta classe.
- ***Socket( )***: implementa um socket em ambos os lados da comunicação. Principais métodos :
  - getInetAddress() : retorna o endereço ao qual o socket está conectado
  - getInputStream() : retorna um stream de entrada para este *socket*
  - getLocalAddress() : obtem o endereço local ao qual o socket está associado
  - getLocalPort() : retorna a porta local do socket
  - getOutputStream() : retorna um stream de saída para o socket.

Um servidor em JAVA

```
import java.io.*;
  import java.net.*;
  import java.awt.*;
  import java.awt.event.*;
  class FechaJanelaETermina extends WindowAdapter {
     public void windowClosing(WindowEvent e)
     { System.exit(0);    }
  }


  class Servidor extends Frame {
     private TextArea mostrador;
     public Servidor() {  // Construtor da classe Servidor
        super ("   Servidor"); // cria o frame do servidor
        mostrador = new TextArea("", 0, 0,
  TextArea.SCROLLBARS_VERTICAL_ONLY);
        add( mostrador, BorderLayout.CENTER);
        setSize(300, 150);
        setVisible(true);
     }
```

Servidor em JAVA (cont)

```
public void ExecutaServidor (){
  ServerSocket servidor;
  Socket conexao;
  DataOutputStream saida;
  DataInputStream entrada;
  int contador = 1;
  try {
     // Passo 1 : criar um servidor de sockets
     servidor = new ServerSocket(2000, 2);//porta 200 até 2 conex
     while (true) {

        // passo2: aceita uma conexao
        conexao = servidor.accept();
        mostrador.append("Conexao "+contador+" recebida de
          "+conexao.getInetAddress().getHostName());

        // passo 3: cria os Streams de entrada e de saida
        entrada = new DataInputStream(conexao.getInputStream());
        saida = new DataOutputStream(conexao.getOutputStream());

        // passo 4: Trata a conexao
        mostrador.append("Enviando msg \"Conectado c/sucesso\"\n");
        saida.writeUTF("Conexao com sucesso");
        mostrador.append(" Msg do cliente ="+entrada.readUTF());

        // passo 5: fechar conexao
        mostrador.append("\nTransm encerrada.FechandoSocket\n\n");
              conexao.close();
              ++contador;
     }
} // fim do try
```

Servidor JAVA (final)

```
        catch(IOException e) {
              e.printStackTrace();
           }
    }  // fim do método
 } // Fim da classe Servidor


public class TesteSocketServidor {
    public static void main(String argv[]) {
       Servidor s = new Servidor();
       s.addWindowListener(new FechaJanelaETermina());
       s.ExecutaServidor();
    }
 }
```

Um cliente em JAVA

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

  class FechaJanelaETermina extends WindowAdapter {
     public void windowClosing(WindowEvent e)
     {
        System.exit(0);
     }
  }


  class Cliente extends Frame {
     private TextArea mostrador;
     public Cliente() {   // Construtor da classe Cliente
        super ("   Cliente"); // cria o frame do cliente
        mostrador = new TextArea("", 0, 0,
            TextArea.SCROLLBARS_VERTICAL_ONLY);
        add( mostrador, BorderLayout.CENTER);
        setSize(300, 150);
        setVisible(true);
     }
```

Um cliente em JAVA

```
public void ExecutaCliente (){
  Socket cliente;
  DataOutputStream saida;
  DataInputStream entrada;
  try {
     // Passo 1 : criar um socket para fazer conexao
     cliente = new Socket(InetAddress.getByname(”feijoada"),200);
     // procura servidor no nó ”feijoada",porta 200

     // passo 2: pega Streams de entrada e de saida
     entrada = new DataInputStream(cliente.getInputStream());
     saida = new DataOutputStream(cliente.getOutputStream());
     mostrador.append("  Streams de E/S criadas \n");

     // passo 3: Trata a conexao
     mostrador.append(" Msg do servidor : "+
            entrada.readUTF()+"\n");
     mostrador.append(" Enviando mensagem: \" Obrigado\"\n");

     saida.writeUTF("Obrigado.");
     // passo 4: fechar conexao
     mostrador.append("\n Transm.encerrada. Fechando Socket\n\n");
     cliente.close();
  }
```

Um cliente em JAVA

```
   catch(IOException e) {
       e.printStackTrace();
     }
}
  } // Fim da classe Cliente
  public class TesteSocketCliente {
     public static void main(String argv[]) {
        Cliente c = new Cliente();
        c.addWindowListener(new FechaJanelaETermina());
        c.ExecutaCliente();
     }
  }
```