Universidade Federal de Santa Catarina
Departamento de Informática e de Estatística

# Capítulo 4
# Camada de Aplicação

**Prof. Roberto Willrich**
**INE - UFSC**
**willrich@inf.ufsc.br**

# API

- Interface para o Programa de Aplicação (API – *Application Program Interface*)
  - É uma interface disponível para programadores
  - Disponibilidade depende tanto do sistema operacional quando da linguagem de programação usados

# Programação via Sockets

**Meta:** ensinar como contruir aplicações cliente/servidor que se comunicam usando sockets

## Socket API

- Introduzido no BSD4.1 UNIX, 1981
- Seguem o paradigma cliente/servidor
- Dois tipos de serviço de transporte via socket API:
  - Datagrama não confiável
  - Confiável e orientado a fluxo de bytes

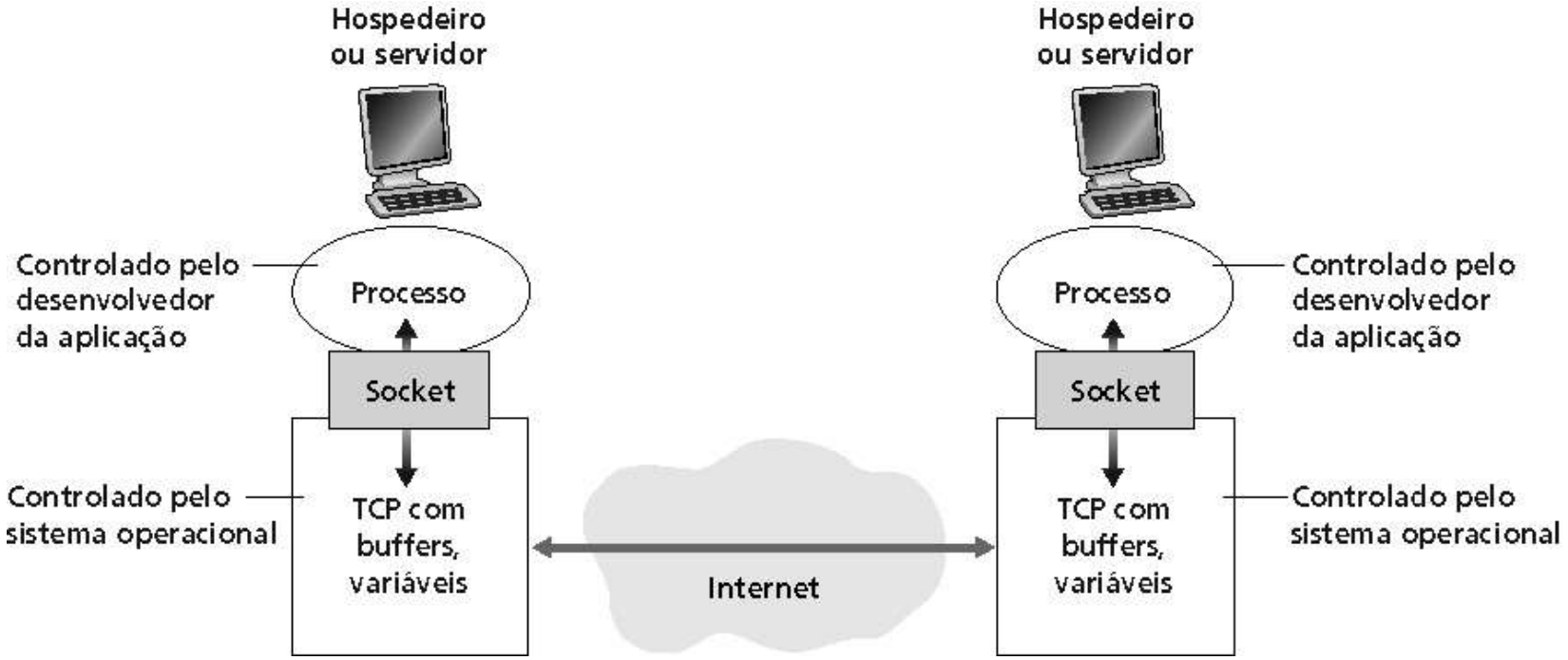
## socket

Uma interface controlada pelo S.O., local ao host, que é criada e controlada pela aplicação
Uma porta na qual o processo de aplicação pode enviar e receber dados de outros processo de aplicação

# Programação de Socket Usando TCP

**Socket:** uma porta entre o processo de aplicação e o protocolo de transporte fim-a-fim (TCP ou UDP)

**Serviço TCP:** transferência confiável de bytes de um processo para outro



# Programação com sockets usando TCP

Cliente deve contactar servidor

- processo servidor deve antes estar em execução
- servidor deve antes ter criado socket (porta) que aguarda contato do cliente

Cliente cria socket TCP:

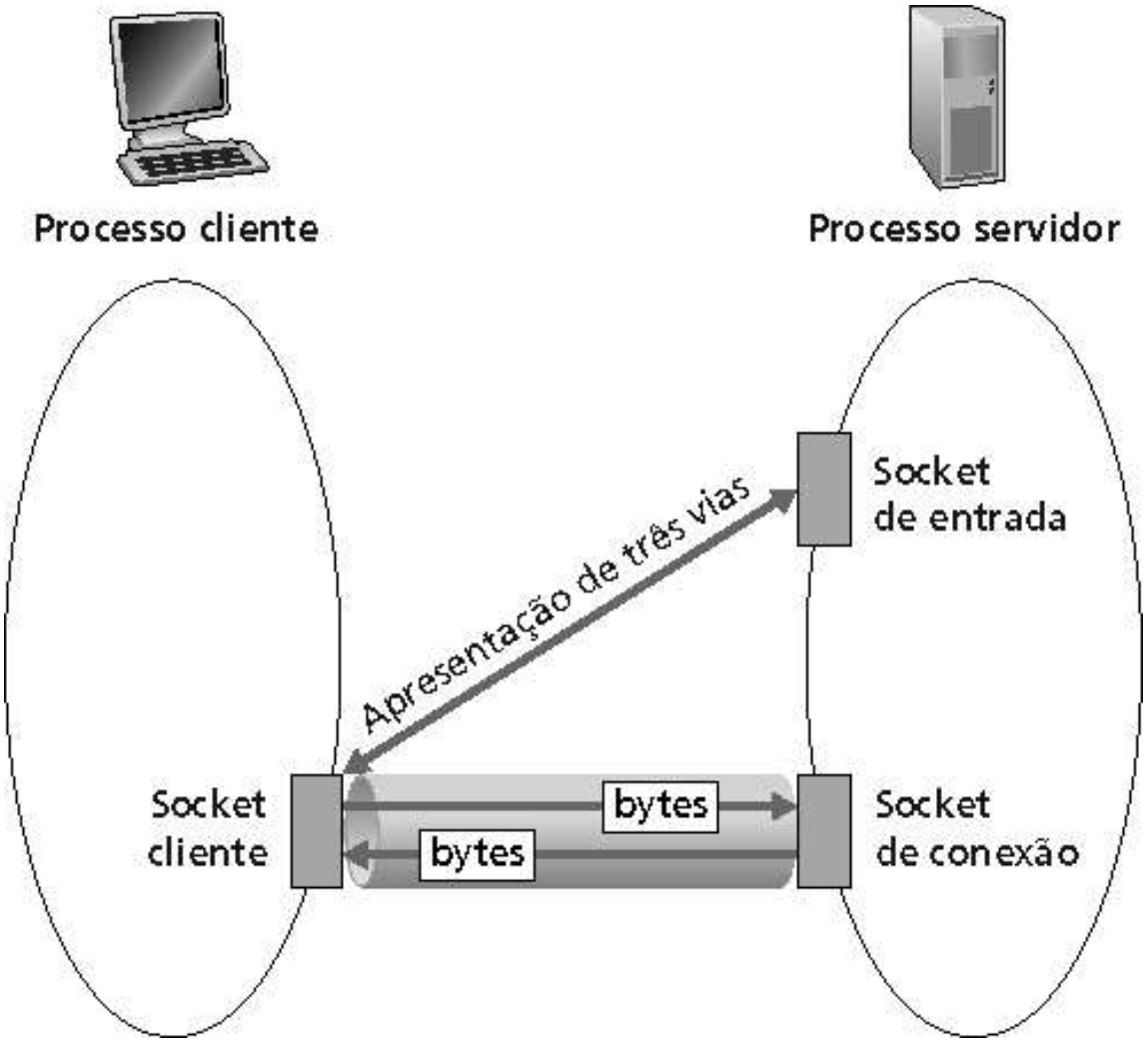
- socket TCP local ao cliente
- especificar endereço IP, número de porta do processo servidor

- Quando cliente cria socket: TCP do cliente estabelece conexão com TCP do servidor
- Quando contatado pelo cliente, o TCP do servidor cria socket novo para que o processo servidor possa se comunicar com o cliente
  - permite que o servidor converse com múltiplos clientes

ponto de vista da aplicação

*TCP provê transferência confiável, ordenada de bytes ("tubo") entre cliente e servidor*

# Comunicação entre sockets

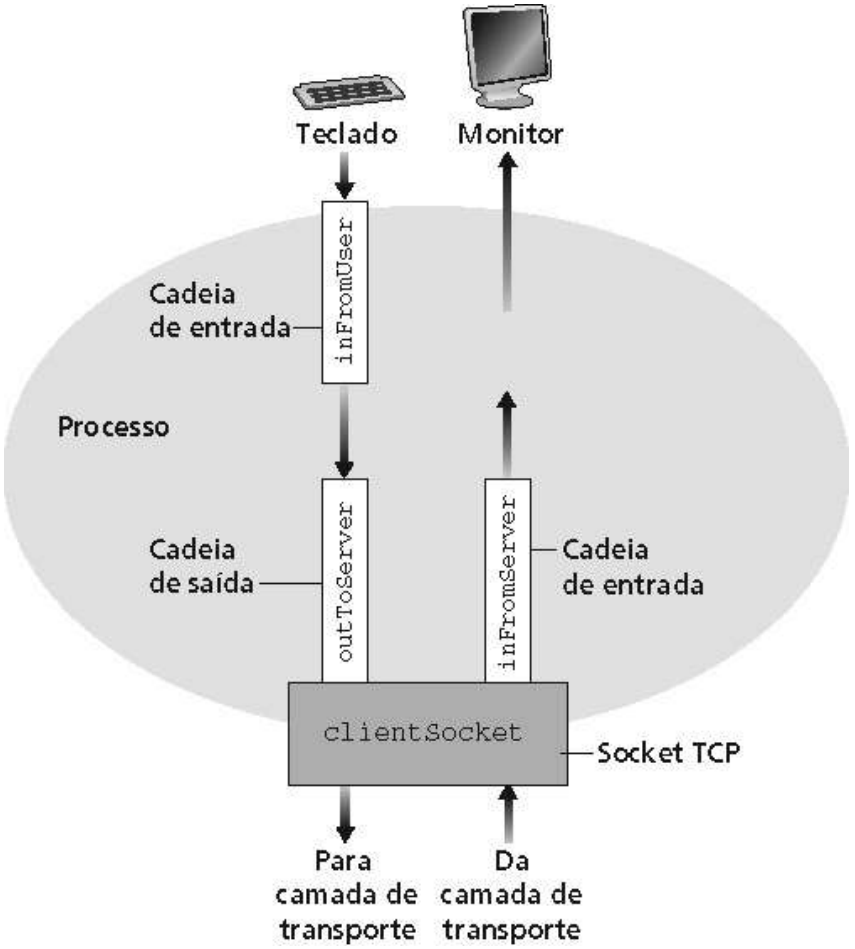


# Jargão stream

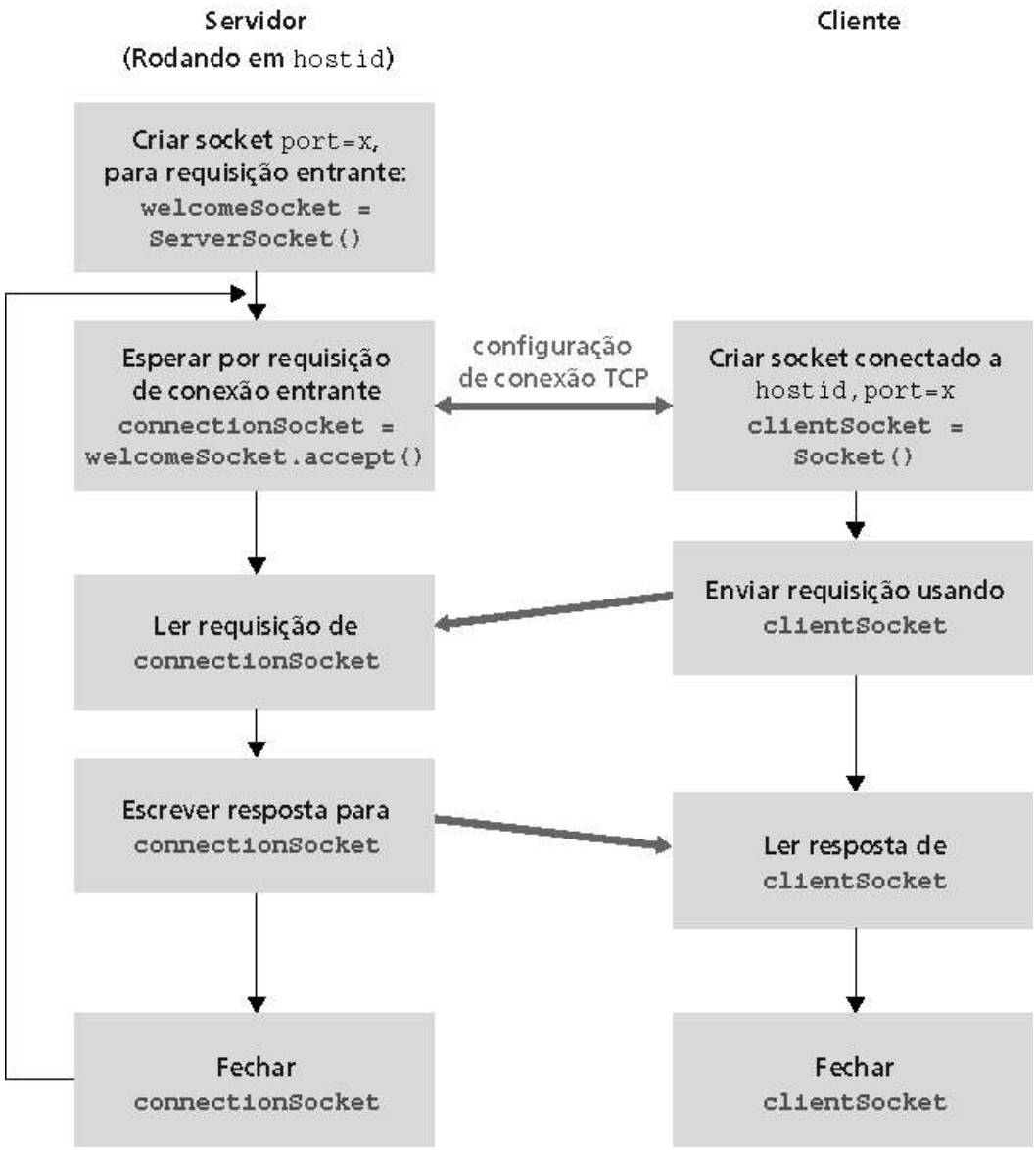
- Um stream é uma seqüência de caracteres que fluem para dentro ou para fora de um processo
- Um stream de entrada é agregado a alguma fonte de entrada para o processo, ex.: teclado ou socket
- Um stream de saída é agregado a uma fonte de saída, ex.: monitor ou socket

# Exemplo de aplicação cliente-servidor

- Cliente lê linha da entrada padrão (fluxo **`inFromUser`**), envia para servidor via socket (fluxo **`outToServer`**)
- servidor lê linha do socket
- servidor converte linha para letras maiúsculas, devolve para o cliente
- cliente lê linha modificada do socket (fluxo **`inFromServer`**), imprime-a



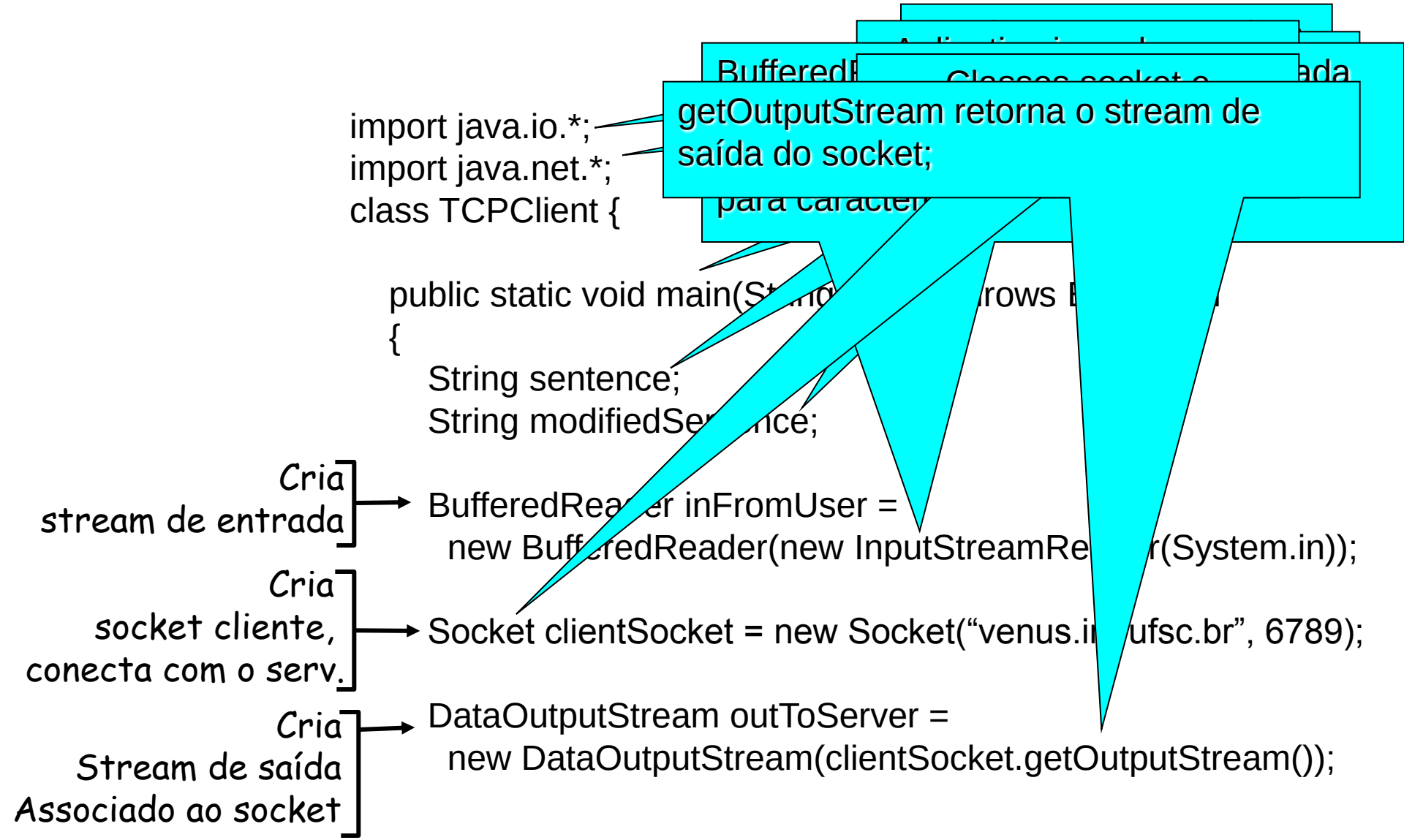
# Interação socket Cliente/servidor: TCP



# Exemplo: cliente Java (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("venus.inf.ufsc.br", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

# Exemplo: cliente Java (TCP)



# Exemplo: cliente Java (TCP)

Cria stream de entrada associado ao socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Envia linha para o servidor

```
outToServer.writeBytes(sentence + '\n');
```

Lê linha do servidor

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

# Exemplo: Servidor Java (TCP)

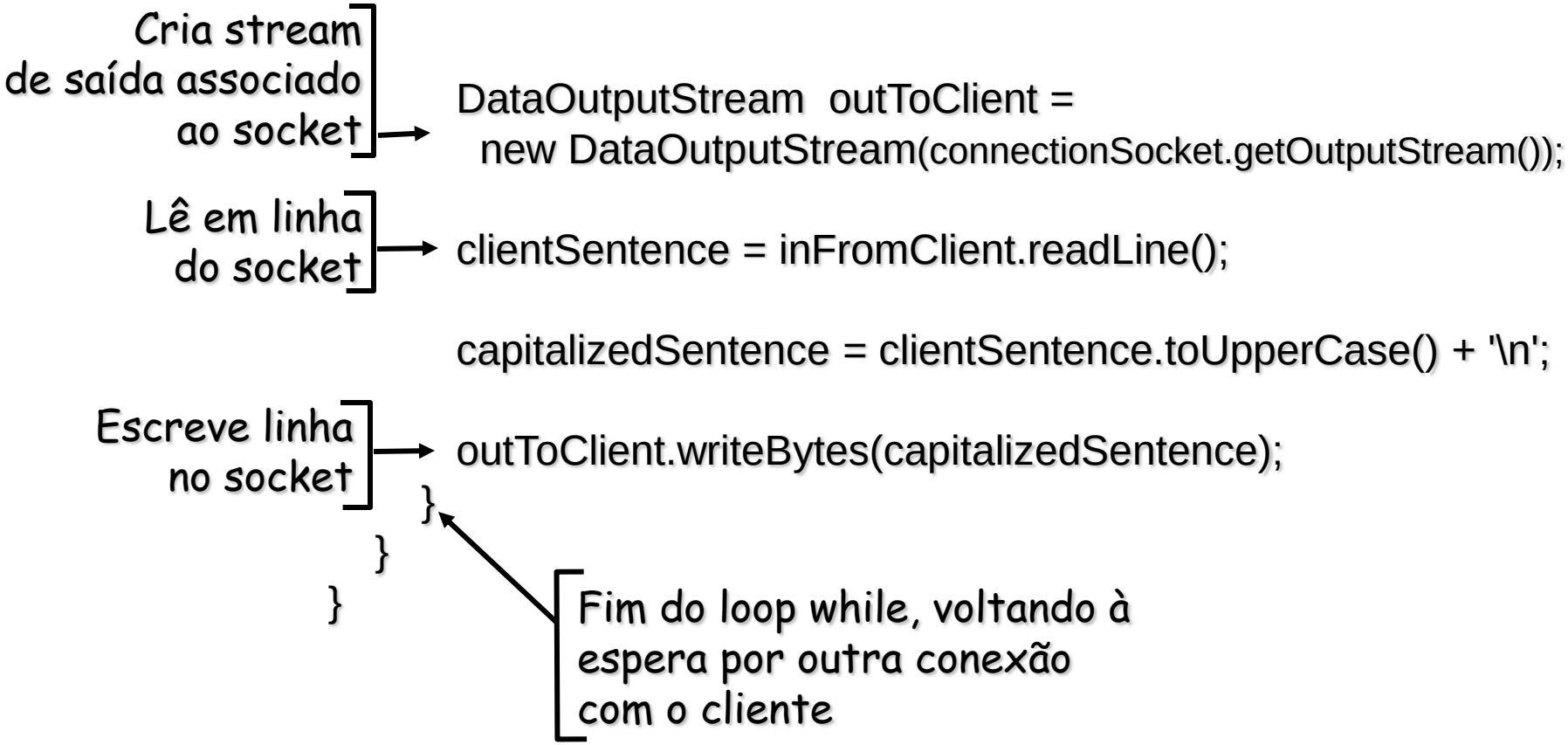
```
import java.io.*;
import java.net.*;

class TCPServer {
  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

       BufferedReader inFromClient =
          new BufferedReader(new
          InputStreamReader(connectionSocket.getInputStream()));
```

Cria socket de escuta na porta 6789

Espera, no socket contatos de clientes

Cria stream de entrada associado ao socket

# Exemplo: Servidor Java (TCP)



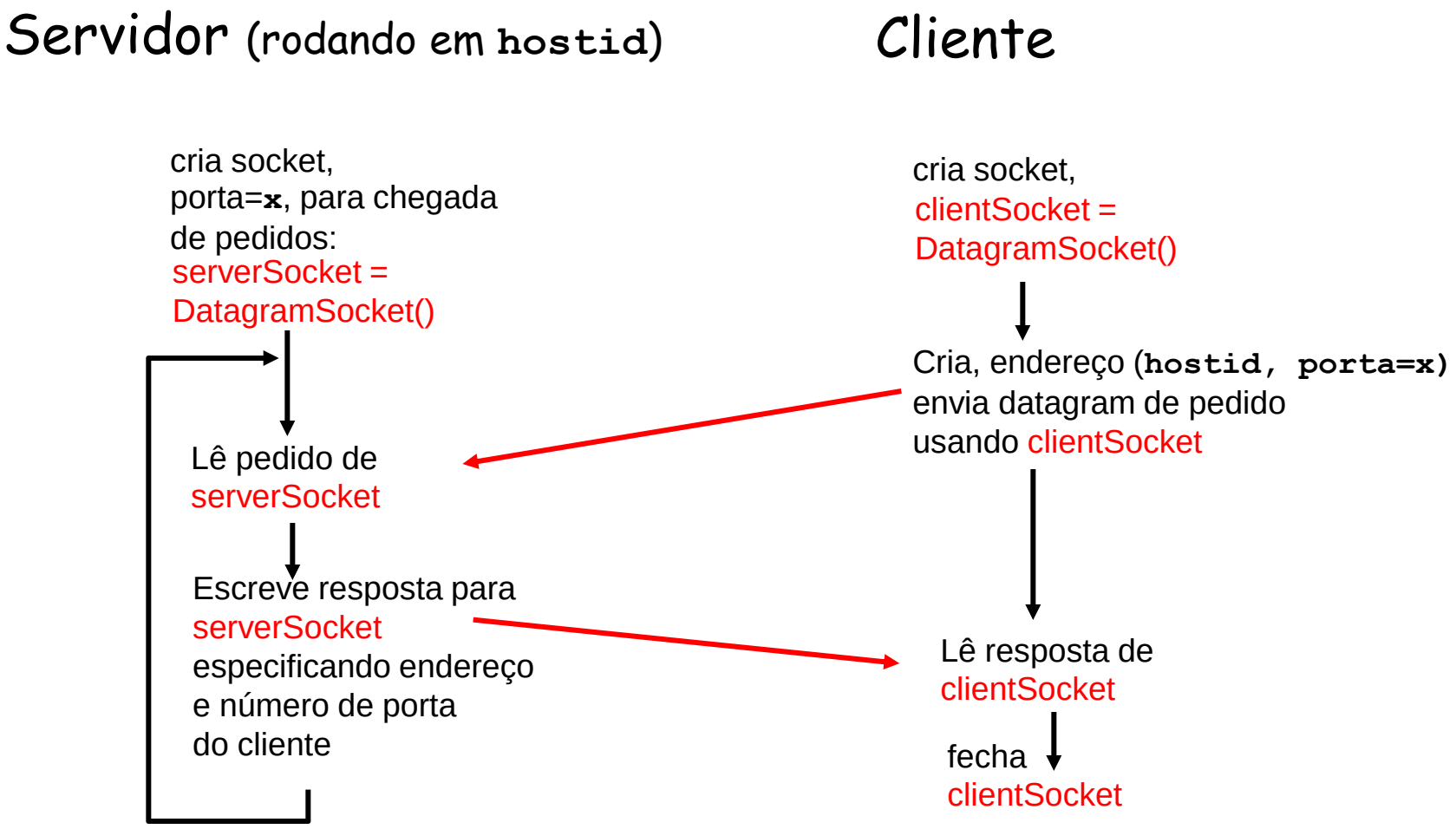
# Programação via socket com UDP

- Comunicação sem "conexão" entre cliente e servidor
  - Emissor associa explicitamente endereço IP e porta do destino
  - Servidor deve extrair endereço IP e porta do emissor do datagrama recebido
- Dado transmitido pode ser recebido fora de ordem ou perdido

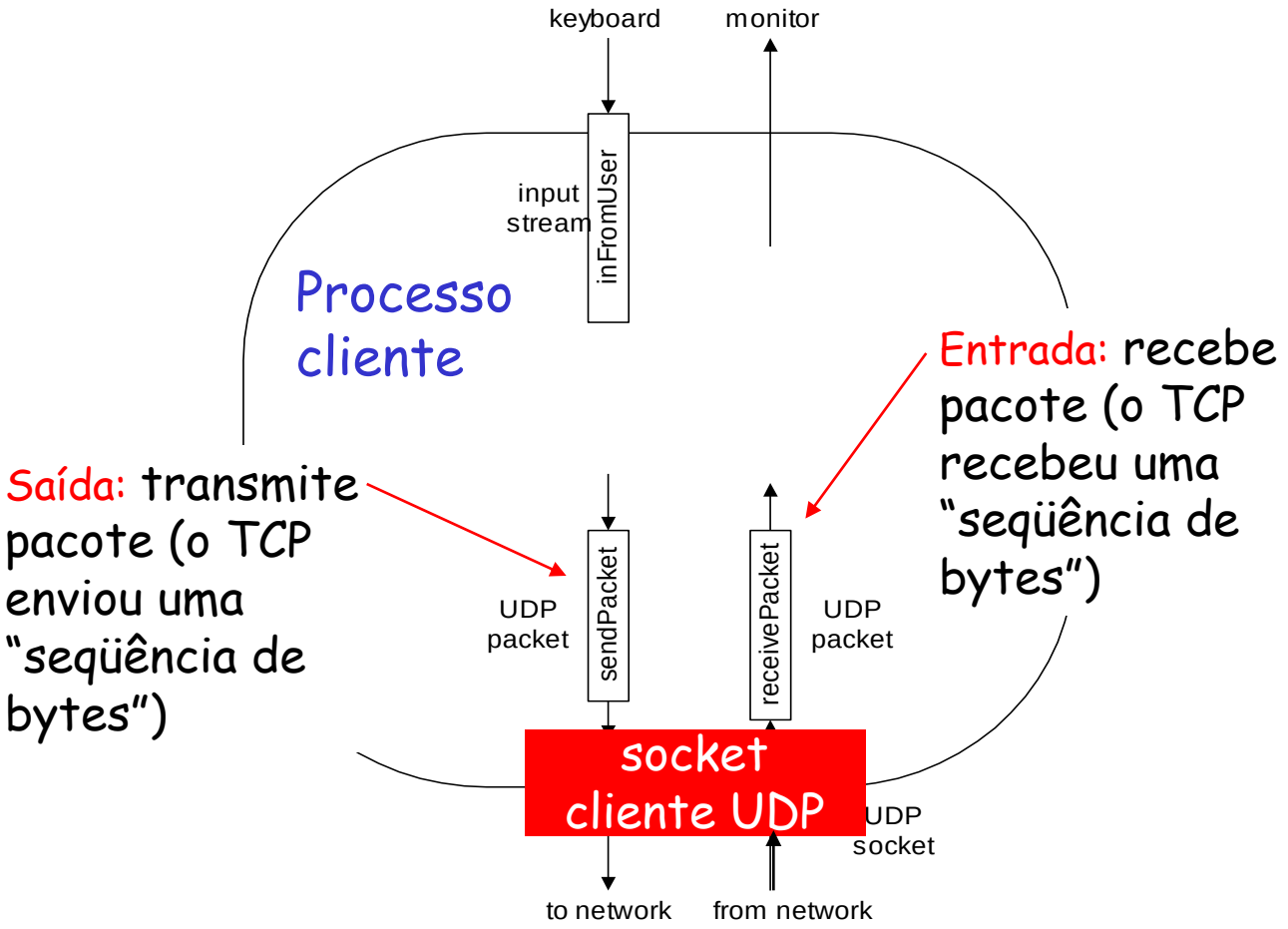
Ponto de vista da aplicação

*UDP fornece transferência não confiável de grupos de bytes entre cliente e servidor*

# Interação socket Cliente/servidor: UDP



# Exemplo: Cliente Java (UDP)



# Exemplo: cliente Java (UDP)

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

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

Cria stream de entrada →

```
BufferedReader inFromUser =
  new BufferedReader(new InputStreamReader(System.in));
```

Cria socket cliente →

```
DatagramSocket clientSocket = new DatagramSocket();
```

Translada hostname para endereço IP usando DNS →

```
InetAddress IPAddress = InetAddress.getByName(
                                    "venus.inf.ufsc.br");

byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];

String sentence = inFromUser.readLine();

sendData = sentence.getBytes();
```

# Exemplo: cliente Java (UDP)

Cria datagrama com dado, tamanho, end. IP, porta

Envia datagrama para o servidor

Lê datagrama do servidor

```
DatagramPacket sendPacket =
new DatagramPacket(sendData, sendData.length,
                                IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
}
```

# Exemplo: Servidor Java (UDP)

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

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

      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {

          DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
```

cria datagram socket no porto 9876

Cria espaço para datagrama recebido

Recebe datagrama

# Exemplo: Servidor Java (UDP)

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
        }
    }
}
```

Obtem end. IP porta, do emissor

Cria datagrama a enviar ao cliente

Escreve datagrama de saída no socket

Fim do loop while,
Volta e espera outros datagramas

# Servidor Web Simples

- Funções do servidor Web:
  - Trata apenas um pedido HTTP por vez
  - Aceita e examina o pedido HTTP
  - Recupera o arquivo pedido do sistema de arquivos do servidor
  - Cria uma mensagem de resposta HTTP consistindo do arquivo solicitado precedido por linhas de cabeçalho
  - Envia a resposta diretamente ao cliente.

# Servidor Web Simples

Contém a classe StringTokenizer que é usada para examinar o pedido

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

class WebServer {
   public static void main(String argv[]) throws Exception
   {
      String requestMessageLine;
      String fileName;

      ServerSocket listenSocket = new ServerSocket(80);
      Socket connectionSocket = listenSocket.accept();

      BufferedReader inFromClient =
         new BufferedReader(new InputStreamReader(
               connectionSocket.getInputStream()));

      DataOutputStream outToClient =
         new DataOutputStream(
               connectionSocket.getOutputStream());
```

Primeira linha da mensagem de pedido HTTP e Nome do arquivo solicitado

Aguarda conexão do cliente

Cria fluxo de Entrada

Cria fluxo de Saída

# Servidor Web Simples, cont

Lê a primeira linha do pedido HTTP que deveria ter o seguinte formato: GET file_name HTTP/1.0

```
requestMessageLine = inFromClient.readLine();
```

Examina a primeira linha da mensagem para extrair o nome do arquivo

```
StringTokenizer tokenizedLine =
   new StringTokenizer(requestMessageLine);
if (tokenizedLine.nextToken().equals("GET")){
   fileName = tokenizedLine.nextToken();
   if (fileName.startsWith("/") == true )
         fileName = fileName.substring(1);

   File file = new File(fileName);
   int numOfBytes = (int) file.length();
```

Associa o fluxo inFile ao arquivo fileName

```
   FileInputStream inFile = new FileInputStream (
         fileName);
```

Determina o tamanho do arquivo e constrói um vetor de bytes do mesmo tamanho

```
   byte[] fileInBytes = new byte[];
   inFile.read(fileInBytes);
```

# Servidor Web Simples, cont

Inicia a construção da mensagem de resposta

Transmissão do cabeçalho da resposta HTTP.

```
                outToClient.writeBytes(
                        "HTTP/1.0 200 Document Follows\r\n");

                if (fileName.endsWith(".jpg"))
                    outToClient.writeBytes("Content-Type: image/jpeg\r\n");
                if (fileName.endsWith(".gif"))
                    outToClient.writeBytes("Content-Type:
                            image/gif\r\n");
                outToClient.writeBytes("Content-Length: " + numOfBytes +
                            "\r\n");
                outToClient.writeBytes("\r\n");

                outToClient.write(fileInBytes, 0, numOfBytes);
                connectionSocket.close();
                }

            else System.out.println("Bad Request Message");
            }
        }
```