Comunicação em Rede

Por Windson Viana

Comunicação entre processos

A forma mais simples é uso de APIs para acessar a camada de sockets do sistema operacional

- Problemas inerentes a desconexões, não presença e protocolo de troca de informação ficam a cargo do programador

Modelos de conexão atrelados ao protocolo de transporte

- Socket TCP, socket UDP
- Uso de Multicast ou Unicast

Suporte a modelo com conexão permanente e manutenção de estado de conexão

Protocolos de troca de mensagens

- Separadores
- Serialização e deserialização
- XML, JSON

Programação de sockets

API socket

- Introduzida no BSD4.1 UNIX em 1981
- Criada, usada e liberada explicitamente pelas APIs.
- Paradigma cliente-servidor

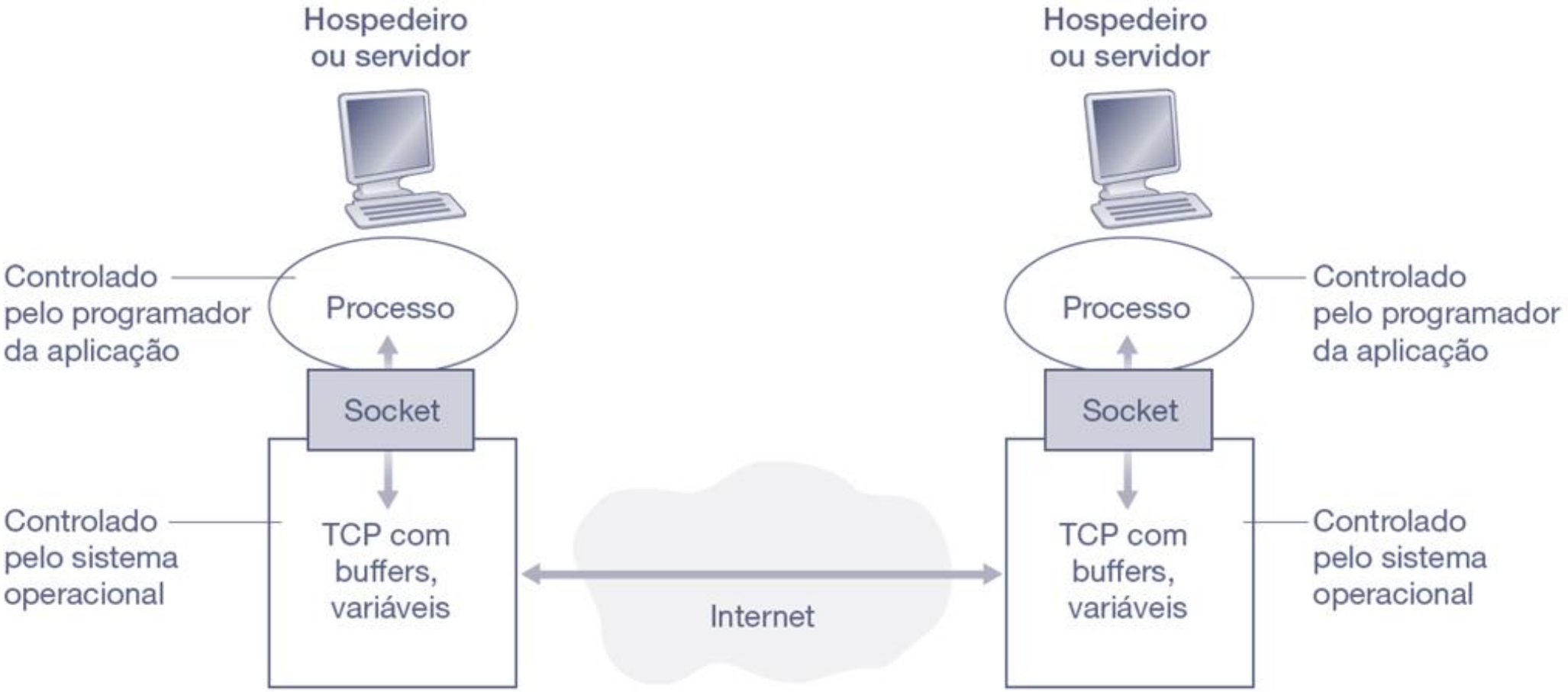
Dois tipos de serviços de transporte por meio da API socket:

- UDP
- TCP

socket

Uma interface criada pela aplicação e controlada pelo SO (uma "porta") na qual o processo da aplicação pode enviar e receber mensagens para/de outro processo da aplicação

Modelo Cliente-Servidor



Modelo Cliente-Servidor

Características

- Uma aplicação de rede consiste em pares de processos que enviam mensagens uns para os outros por meio de uma rede.
- Um processo envia mensagens para a rede e recebe mensagens dela através de uma interface de software denominada socket.

Requisitos

- Para identificar o processo receptor, duas informações devem ser especificadas:
- O endereço do hospedeiro e um identificador que especifica o processo receptor no hospedeiro de destino.

Fundamentos de programação de socket

Servidor

- servidor deve estar rodando antes que o cliente possa lhe enviar algo
- servidor deve ter um socket (porta) pelo qual recebe e envia segmentos
- da mesma forma, o cliente precisa de um socket

Identificação

- socket é identificado localmente com um número de porta
  - semelhante ao número de apartamento de um prédio
- cliente precisa saber o endereço IP do servidor e o número de porta do socket

Programação de socket com UDP

- UDP: sem "conexão" entre cliente e servidor
  - sem "handshaking"
- Emissor conecta de forma explícita endereço IP e porta do destino a cada segmento
  - SO conecta endereço IP e porta do socket emissor a cada segmento
  - Servidor pode extrair endereço IP, porta do emissor a partir do segmento recebido

Ponto de vista da aplicação

- UDP oferece transferência não confiável de grupos de bytes ("datagramas") entre cliente e servidor

Nota: A terminologia oficial para um pacote UDP é "datagrama". Nesta aula, usamos "segmento UDP" em seu lugar.

Exemplo em curso

cliente:

- usuário digita linha de texto
- programa cliente envia linha ao servidor

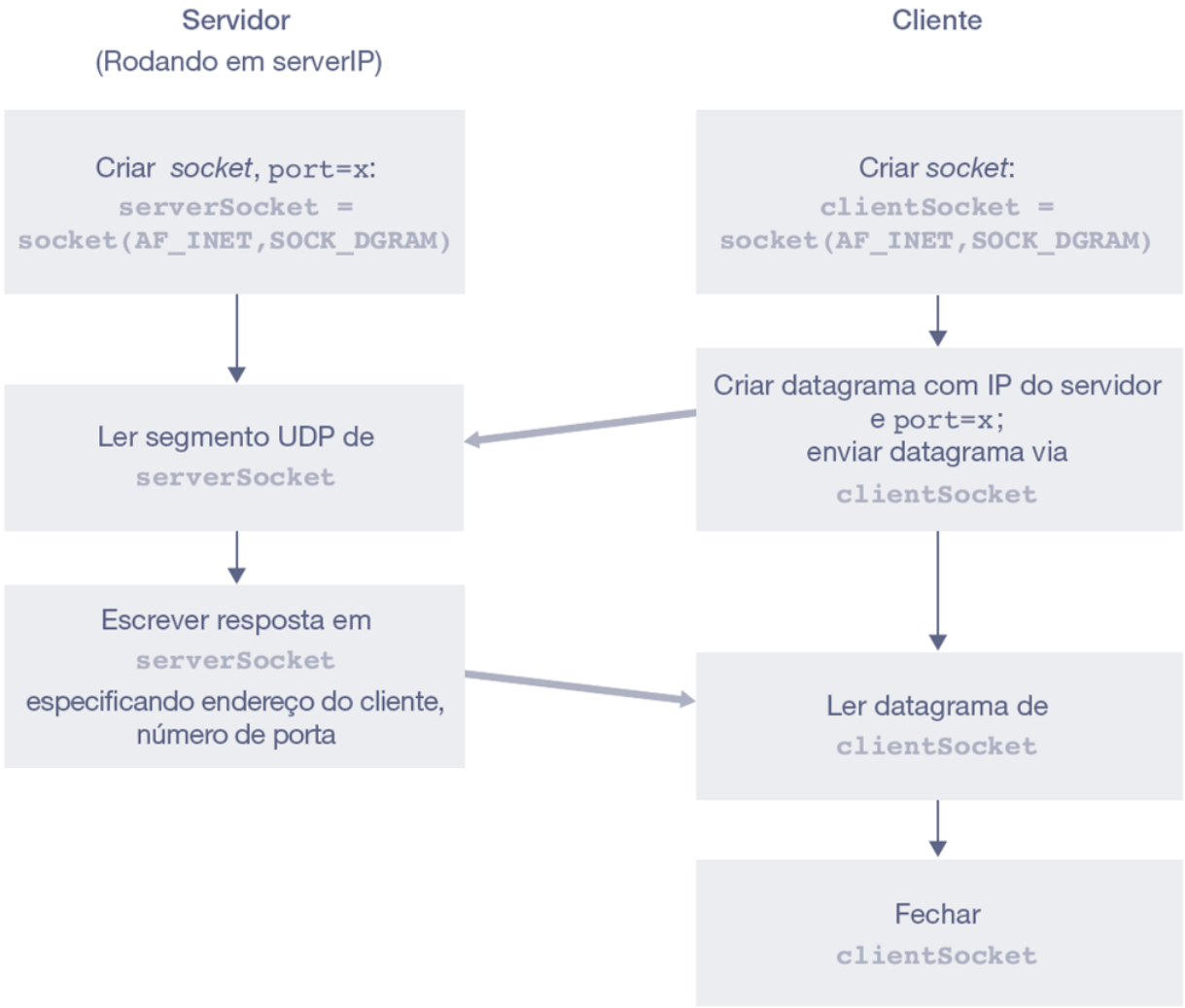
servidor:

- servidor recebe linha de texto
- coloca todas as letras em maiúsculas
- envia linha modificada ao cliente

cliente:

- recebe linha de texto
- apresenta

Programação de sockets com UDP



Código Java do Cliente

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

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

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

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

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

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

cria cadeia de entrada

cria socket do cliente

traduz hostname para endereço IP usando DNS

Código Java do Cliente ...

cria datagrama com dados a enviar, tamanho, end. IP, porta

envia datagrama ao 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();
}

}
```

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 socket de datagrama na porta 9876

cria espaço para datagrama recebido

recebe datagrama

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);
            }
        }

    }
```

obtém end. IP # porta do emissor

cria datagrama p/ enviar ao cliente

escreve datagrama no socket

fim do loop while, retorna e espera outro datagrama

Observações e perguntas sobre UDP

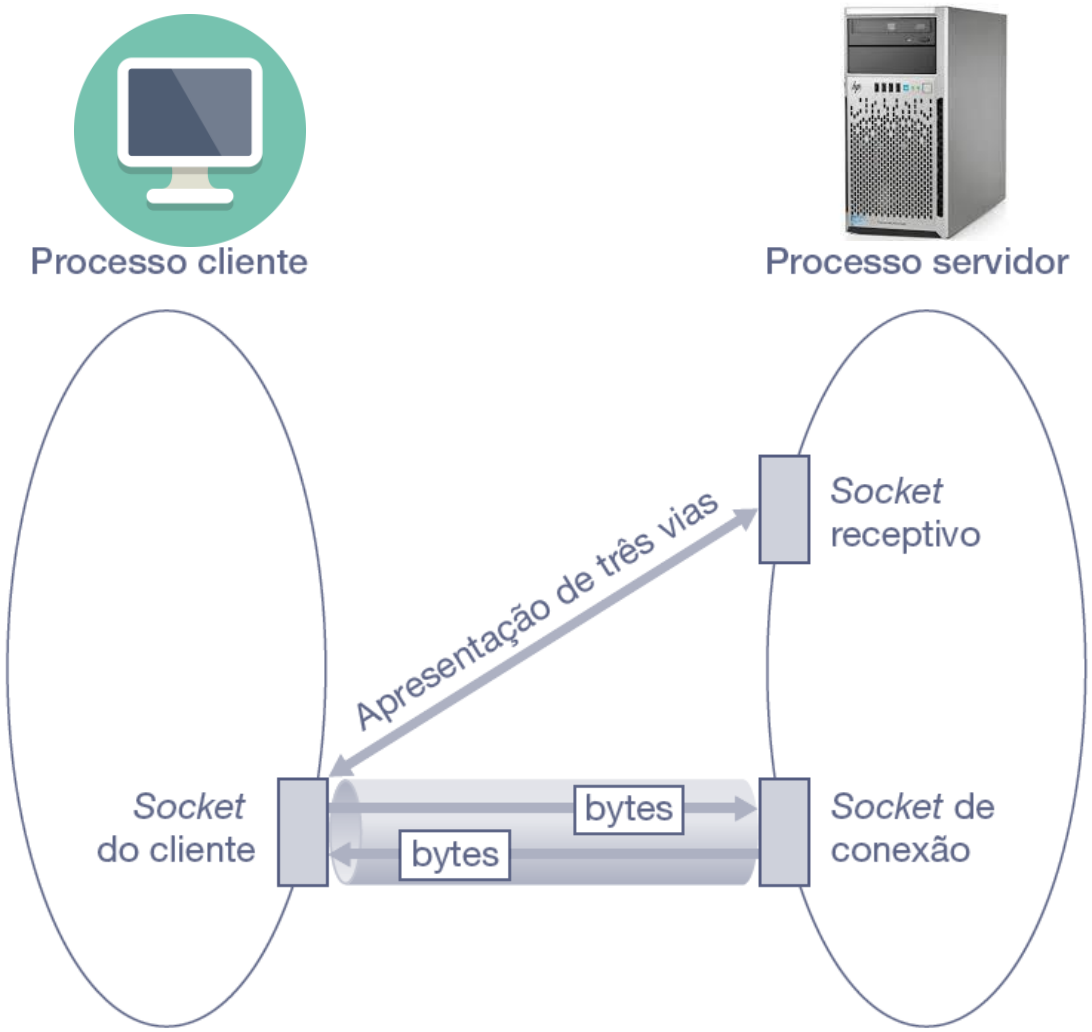
Características

- Cliente e servidor usam DatagramSocket
- IP e porta de destino são explicitamente conectados ao segmento.
- Execute e o código e responda

Perguntas

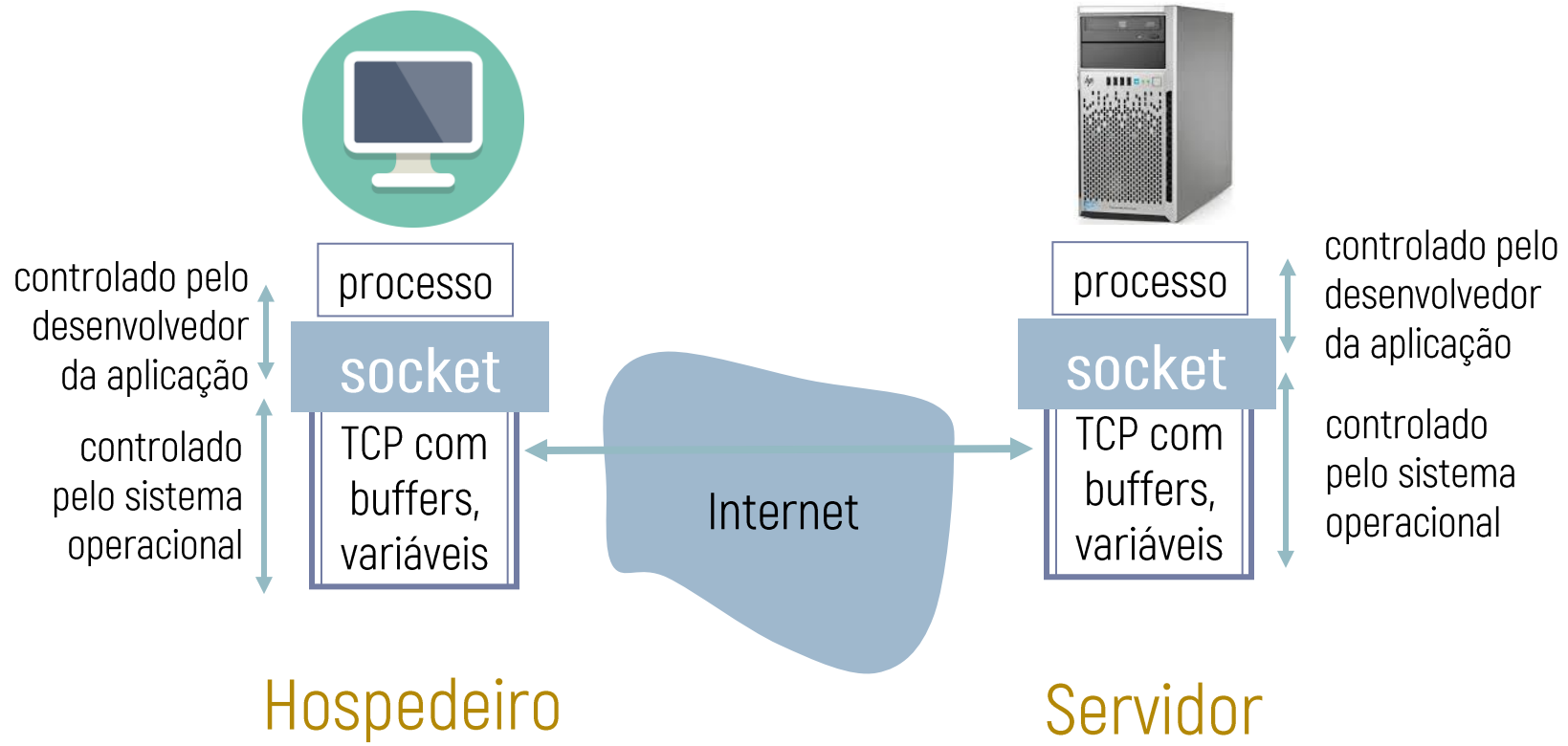
- O que acontece se executarmos o cliente sem o servidor está ativado? Ocorre algum erro?
- O cliente pode enviar um segmento ao servidor sem saber o endereço IP e/ou número de porta do servidor?
- Múltiplos clientes podem usar esse servidor? Teste!

Socket TCP


Processo cliente
Processo servidor
Apresentação de três vias
Socket receptivo
Socket do cliente
bytes
bytes
Socket de conexão

Programação de socket usando TCP

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



Programação de socket com TCP

Cliente deve contactar servidor

- Processo servidor primeiro deve estar rodando
- Servidor deve ter criado um socket (porta) que aceita contato do cliente
- Cliente contacta servidor:
  - criando socket TCP local ao cliente
  - especificando endereço IP, # porta do processo servidor
- Quando cliente cria o socket: cliente TCP estabelece conexão com servidor TCP

Lado Servidor

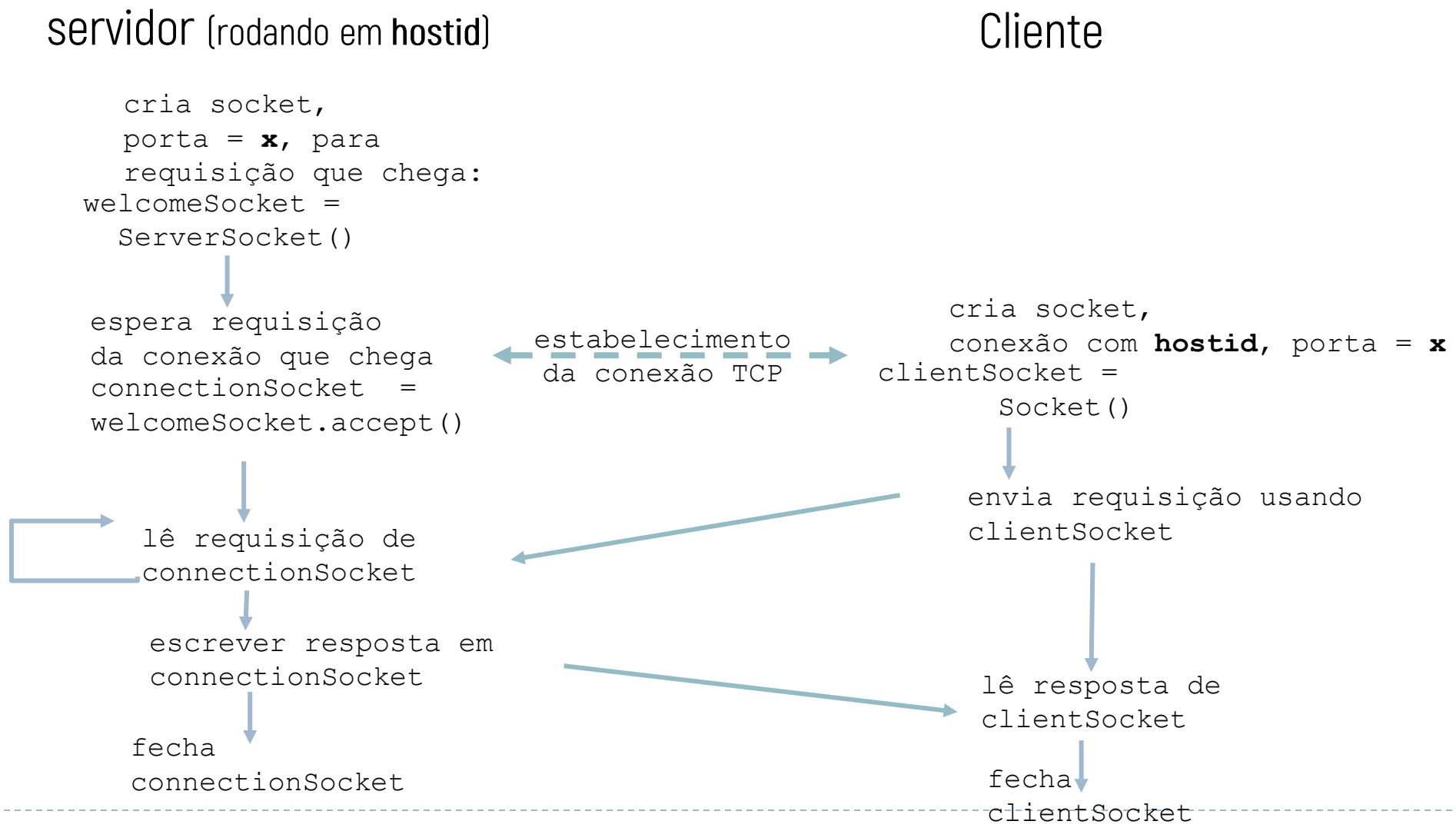
- Quando contactado pelo cliente, servidor TCP cria novo socket para processo servidor se comunicar com cliente
- Permite que servidor fale com múltiplos clientes
  - números de porta de origem usados para distinguir clientes (mais no Cap. 3)

Lembrem!

Ponto de vista da aplicação

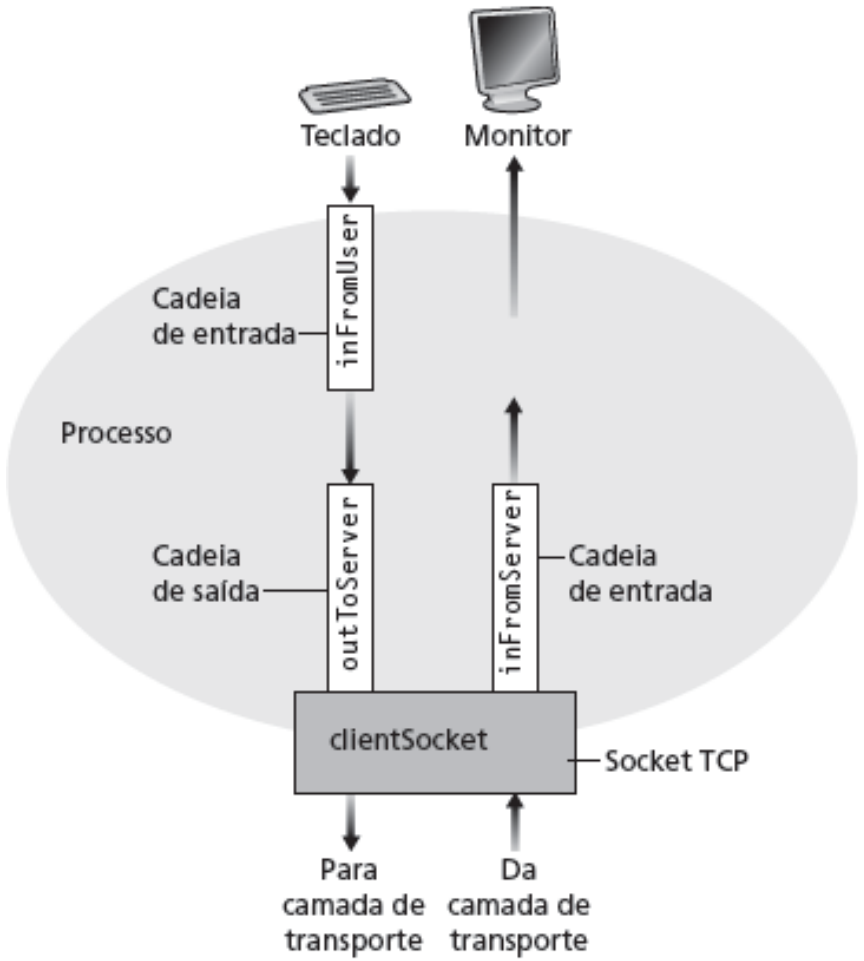
TCP oferece transferência de bytes confiável, em ordem ("pipe") entre cliente e servidor

Interação de socket cliente/servidor: TCP



Jargão de cadeia

- uma cadeia é uma sequência de caracteres que flui para dentro ou fora de um processo.
- uma cadeia de entrada está conectada a uma fonte de entrada para o processo, p. e., teclado ou socket.
- uma cadeia de saída está conectada a uma fonte de saída, p. e., monitor ou socket.



TCP – observações e perguntas

Servidor tem dois tipos de sockets:

- ServerSocket e Socket

Quando o cliente bate na "porta" de serverSocket, servidor cria connectionSocket e completa conexão TCP.

IP de destino e porta não são explicitamente conectados ao segmento.

Múltiplos clientes podem usar o servidor?

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("hostname", 6789);

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

cria cadeia de entrada

cria socket cliente, conexão com servidor

cria cadeia de saída conectada ao socket

Cliente Java (TCP) ...

cria cadeia de entrada conectada ao socket

envia linha ao servidor

lê linha do servidor

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

sentence = inFromUser.readLine();

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

modifiedSentence = inFromServer.readLine();

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

clientSocket.close();

    }
}
```

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 apresentação na porta 6789

espera no socket de apresentação pelo contato do cliente

cria cadeia de entrada, conectada ao socket

Servidor Java (TCP) ...

cria cadeia de saída, conectada ao socket

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

lê linha do socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

escreve linha no socket

```
outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

fim do loop while, retorna e espera outra conexão do cliente

Vamos a prática!