

Programação de Sockets

Objetivo: aprender a construir aplicações cliente/servidor que se comunicam usando sockets

Socket API

- introduzida no BSD4.1 UNIX, 1981
- explicitamente criados, usados e liberados pelas aplicações
- paradigma cliente/servidor
- dois tipos de serviço de transporte via socket API:
 - datagrama não confiável
 - confiável, orientado a cadeias de bytes

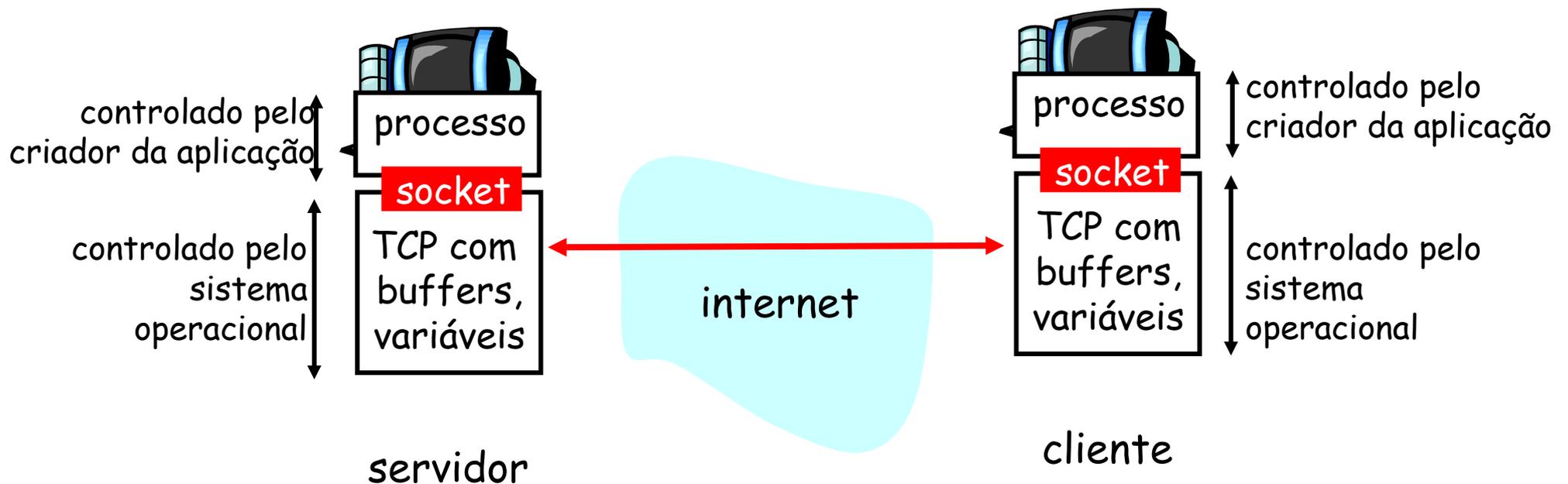
socket

uma interface *local, criada e possuída pelas aplicações, controlada pelo OS* (uma "porta") na qual os processo de aplicação podem *tanto enviar quanto receber* mensagens de e para outro processo de aplicação (local ou remoto)

Programação de Sockets com TCP

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

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



Programação de Sockets *com* *TCP*

Cliente deve contatar o servidor

- processo servidor já deve estar executando antes de ser contactado
- servidor deve ter criado socket (porta) que aceita o contato do cliente

Cliente contata o servidor através de:

- criando um socket TCP local
- especificando endereço IP e número da porta do processo servidor

- Quando o cliente cria o socket cliente TCP estabelece conexão com o TCP do servidor
- Quando contactado pelo cliente, o TCP do servidor cria um novo socket para o processo servidor comunicar-se com o cliente
 - permite o servidor conversar com múltiplos clientes

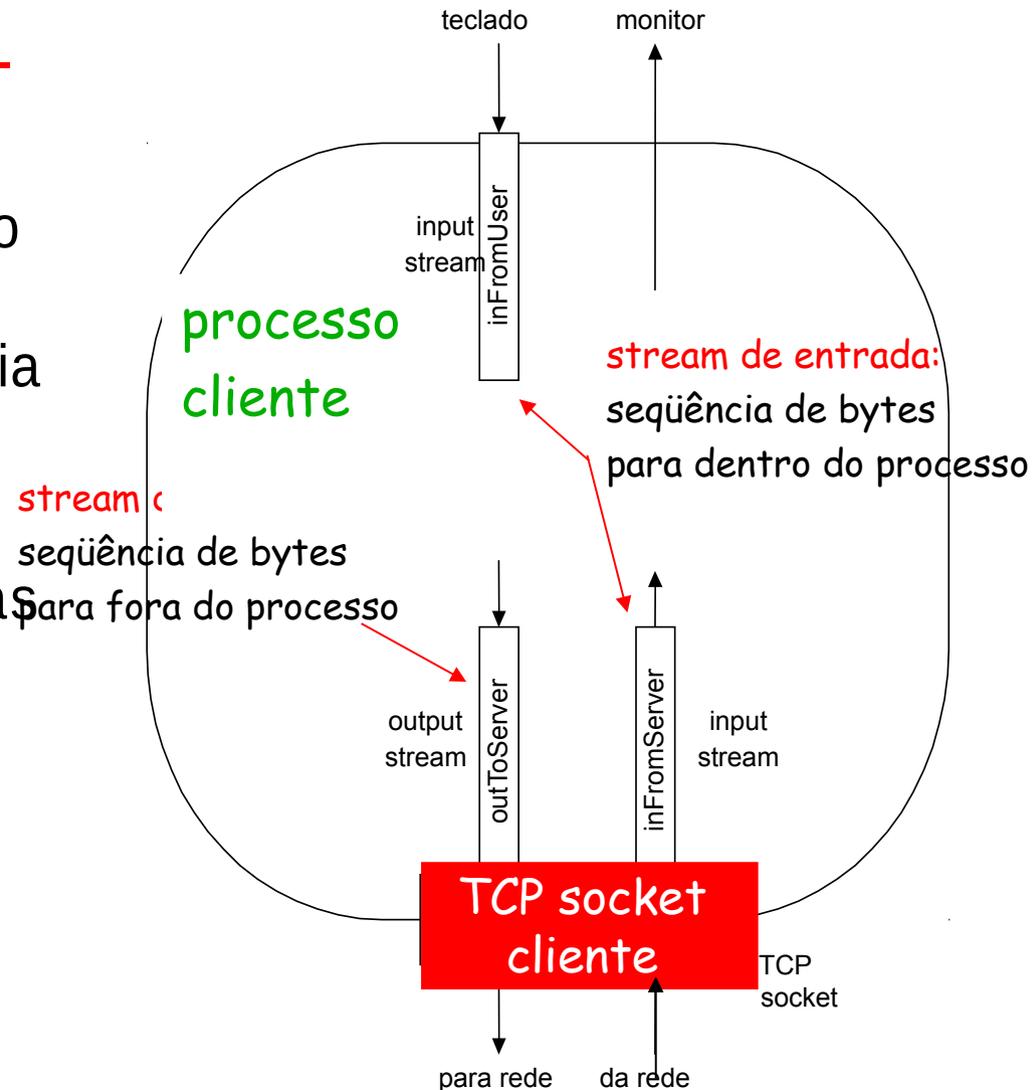
ponto de vista da aplicação

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

Programação de Sockets *com TCP*

Exemplo de aplicação cliente-servidor:

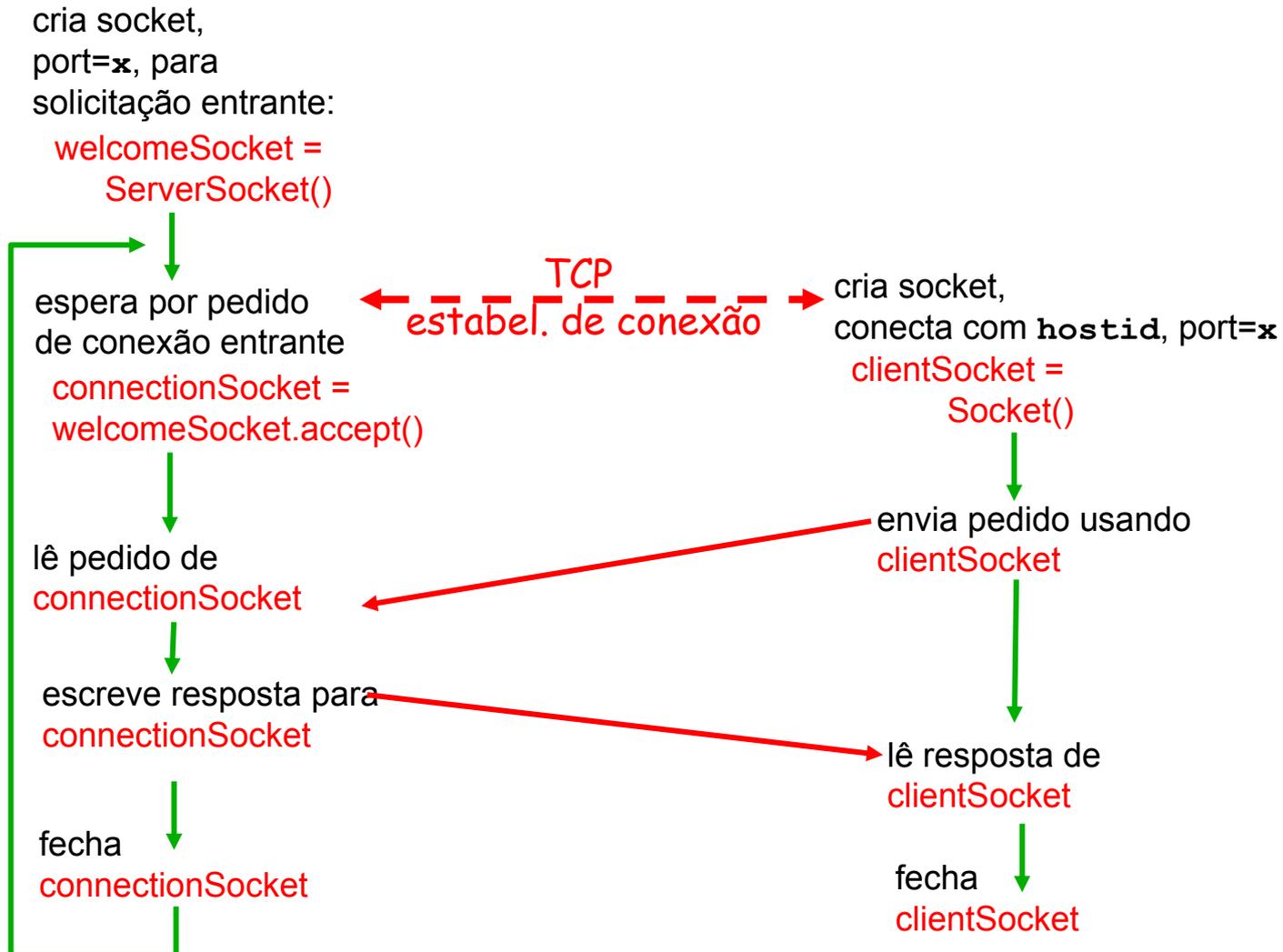
- cliente lê linha da entrada padrão do sistema (`inFromUser` stream), envia para o servidor via socket (`outToServer` stream)
- servidor lê linha do socket
- servidor converte linha para letras maiúsculas e envia de volta ao cliente
- cliente lê a linha modificada através do (`inFromServer` stream)



Interação Cliente/servidor: TCP

Servidor (executando em `hostid`)

Cliente



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("hostname", 6789);
```

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

Cria
stream de entrada

Cria
socket cliente,
conecta ao servidor

Cria
stream de saída
ligado ao socket

Exemplo: cliente Java (TCP), cont.

```
        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();  
  
    }  
}
```

Cria stream de entrada ligado ao socket

Envia linha para o servidor

Lê linha do servidor

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 aceitação
na porta 6789



Espera, no socket
de aceitação por
contato do cliente



Cria stream de
entrada, ligado
ao socket



Exemplo: servidor Java (cont)

Cria stream de
saída, ligado ao
socket

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

Lê linha do
socket

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

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

Escreve linha
para o socket

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

```
}  
}  
}
```

Fim do while loop,
retorne e espere por
outra conexão do cliente

Programação de Sockets *com* *UDP*

UDP: não há conexão entre o cliente e o servidor

- não existe apresentação
- transmissor envia explicitamente endereço IP e porta de destino em cada mensagem
- servidor deve extrair o endereço IP e porta do transmissor de cada datagrama recebido
- **UDP: dados transmitidos podem ser recebidos fora de ordem ou perdidos**

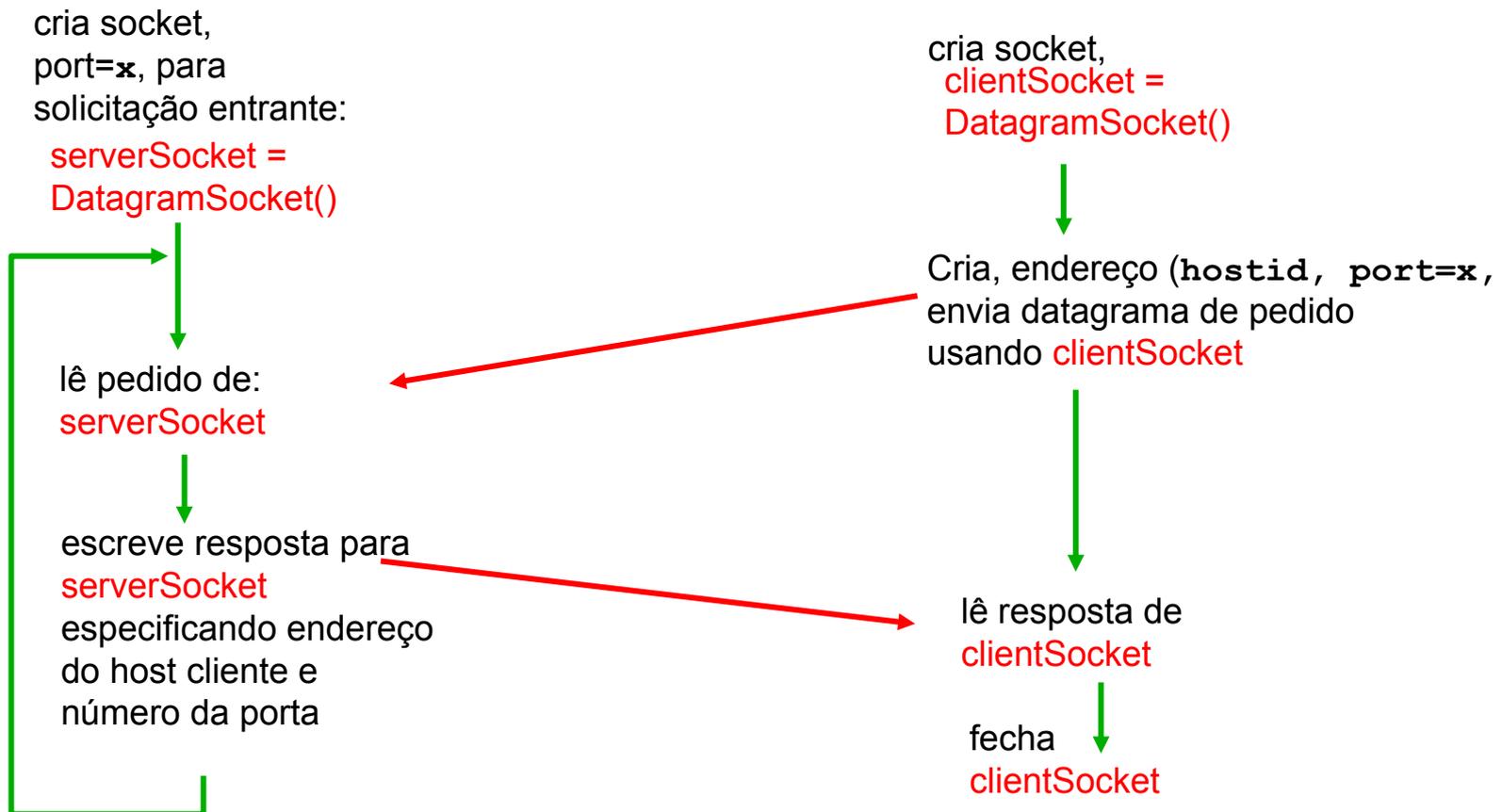
ponto de vista da aplicação

UDP fornece a transferência não confiável de grupos de bytes ("datagramas") entre o cliente e o servidor

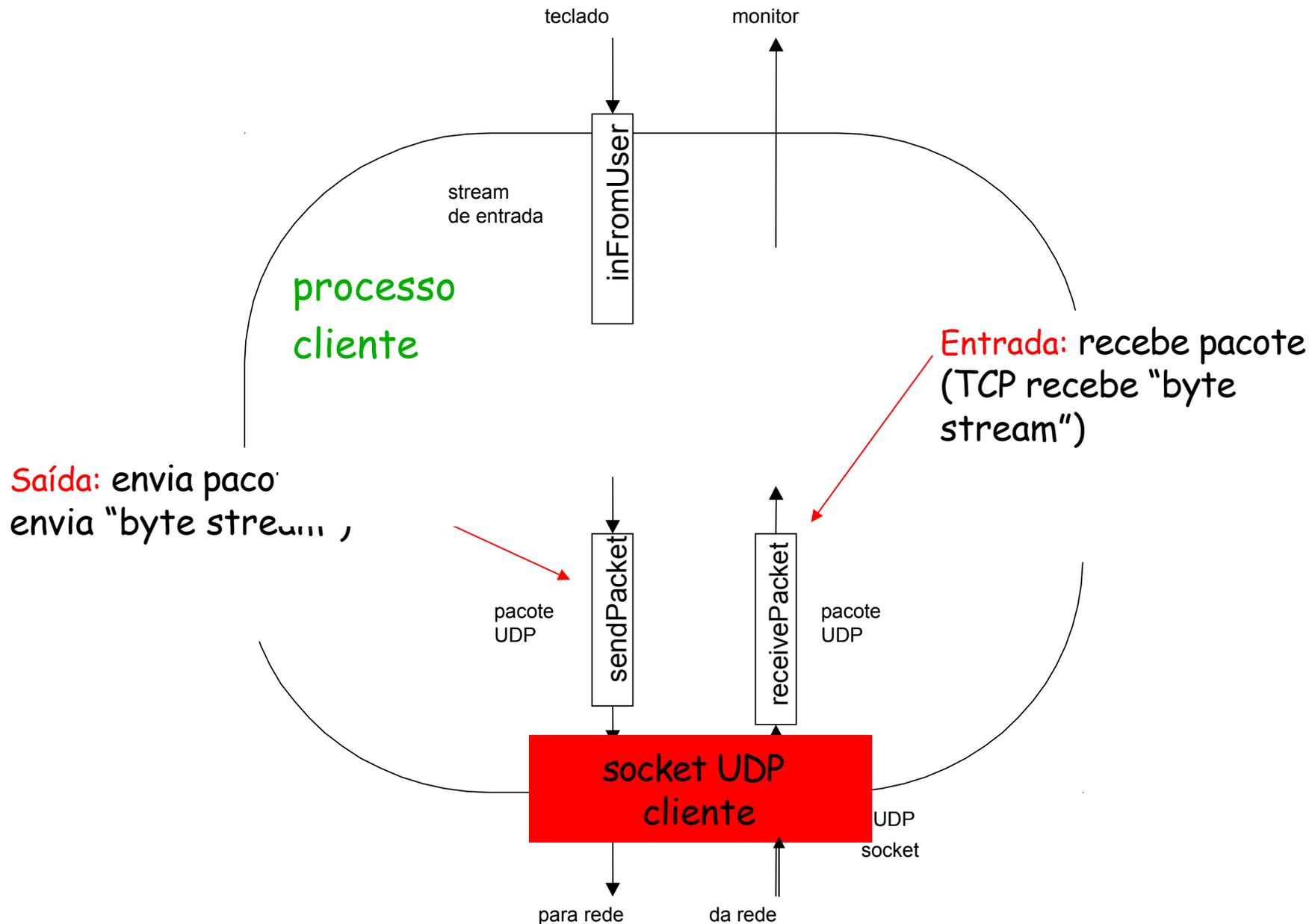
Interação Cliente/servidor: UDP

Servidor (executando `hostid`)

Cliente



Exemplo: cliente Java (UDP)



Exemplo: cliente Java (UDP)

```
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();
    }
}
```

stream de entrada

Cria

socket cliente

Cria

Translada
nome do host para
endereço IP
usando DNS

Exemplo: cliente Java (UDP), cont.

Cria datagrama com
dados a enviar,
tamanho, endereço IP
porta

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

Envia datagrama
para servidor

```
clientSocket.send(sendPacket);
```

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

Lê datagrama
do servidor

```
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
    {
```

Cria
socket datagrama
na porta 9876

```
        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 espaço para
datagramas recebidos

Recebe
datagrama

Exemplo: servidor Java, (cont.)

Obtém endereço IP
e número da porta
do transmissor

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

```
InetAddress IPAddress = receivePacket.getAddress();
```

```
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();
```

```
sendData = capitalizedSentence.getBytes();
```

Cria datagrama
para enviar ao cliente

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

Escreve o
datagrama para
dentro do socket

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

```
}  
}  
}
```

Termina o while loop,
retorna e espera por
outro datagrama

Programação de Sockets: referências

tutorial sobre C-language tutorial (audio/slides):

- “Unix Network Programming” (J. Kurose),
<http://manic.cs.umass.edu>.

Tutoriais sobre Java:

- “Socket Programming in Java: a tutorial,”
<http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html>

Parte 2: Sumário

Nosso estudo das aplicações está agora completo!

- exigências dos serviços de aplicação:
 - confiabilidade, banda passante, atraso
- paradigma cliente-servidor
- modelo do serviço de transporte da Internet I
 - orientado à conexão, confiável: TCP
 - não confiável, datagramas: UDP
- protocolos específicos:
 - http
 - ftp
 - smtp, pop3
 - dns
- programação de sockets
 - implementação cliente/servidor
 - usando sockets tcp, udp

Parte 2: Sumário

Mais importante: características dos *protocolos*

- típica troca de mensagens comando/resposta:
 - cliente solicita informação ou serviço
 - servidor responde com dados e código de status
 - formatos das mensagens:
 - cabeçalhos: campos que dão informações sobre os dados
 - dados: informação sendo comunicada
- controle vs. dados
 - in-band, out-of-band
- centralizado vs. descentralizado
- stateless vs. stateful
- transferência de mensagens confiável vs. não confiável
- “complexidade na borda da rede”
- segurança: autenticação