Programação de sockets

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

SOCKET = API (Application Program Interface)

- 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

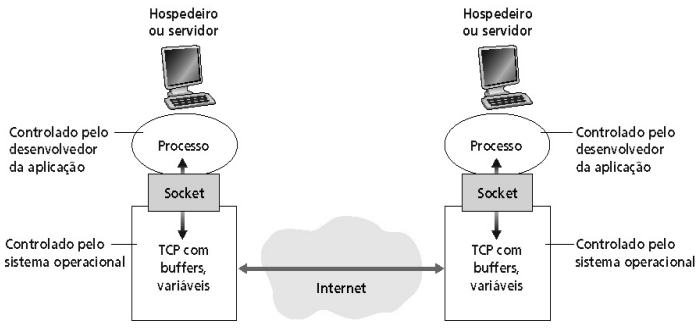
Uma interface **local**, criada por aplicações, **controlada pelo OS** (uma "porta") na qual os **processos** de aplicação podem tanto enviar quanto receber mensagens de e para outro processo de aplicação (local ou remoto)

* Berkeley Software Distribution (BSD, às vezes chamada de Berkeley Unix) é o sistema operacional UNIX derivados desenvolvidos e distribuídos pela Computer Systems Research Group (CSRG), da Universidade da Califórnia, em Berkeley, de 1977 a 1995.

Programação de sockets com TCP

Socket: uma porta entre o processo de aplicação e o protocolo de transporte fim-a-fim (UCP 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 em execução
- Servidor deve ter criado socket (porta) que aceita o contato do cliente

Cliente contata o servidor
- 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

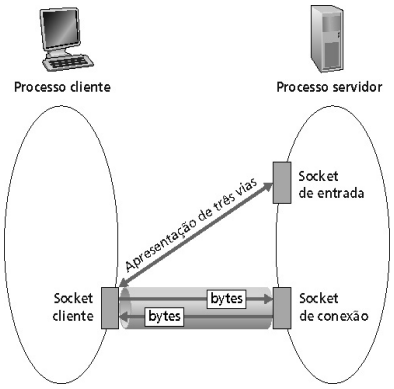
Quando contatado pelo cliente, o TCP do servidor cria um novo socket para o processo servidor comunicar-se com o cliente
- Permite ao servidor conversar com múltiplos clientes
- Números da porta de origem são usados para distinguir o cliente

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

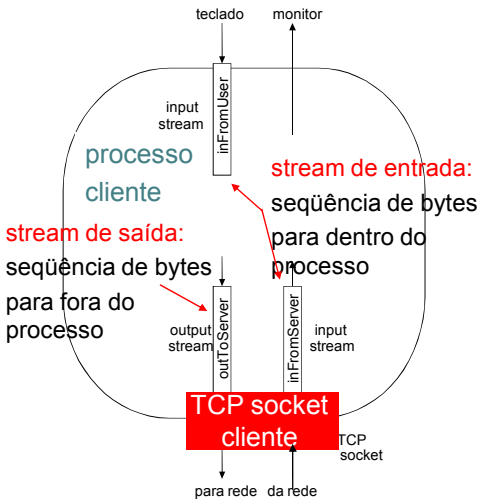
Exemplo de aplicação cliente-servidor:

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

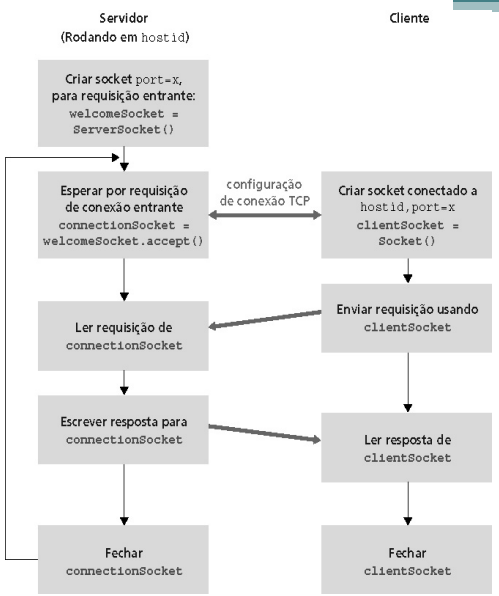


Programação de sockets com TCP

Processo cliente



Interação 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("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)

```
      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 (TCP)

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

       clientSentence = inFromClient.readLine();

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

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

Cria stream de saída ligado ao socket

Lê linha do socket

Escreve linha para o socket

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

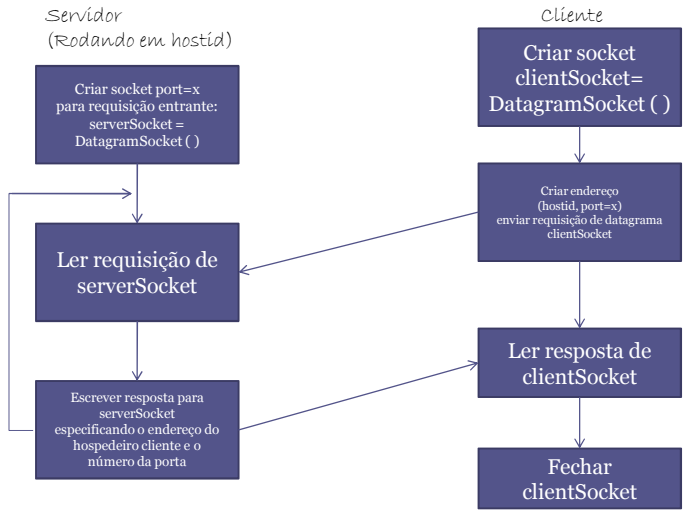
- Não existe apresentação
- Transmissor envia explicitamente endereço IP e porta de destino em cada mensagem
- Servidor deve extrair o endereço IP e porta do transmissor de cada datagrama recebido

UDP: dados transmitidos podem ser recebidos fora de ordem ou perdidos

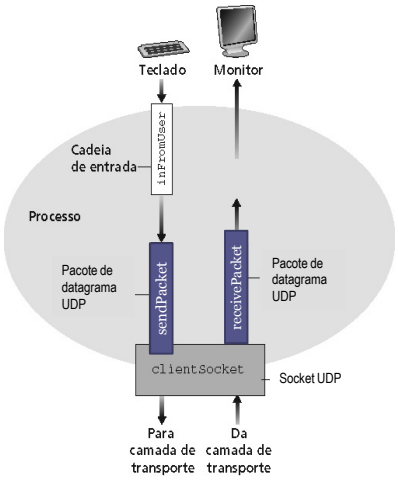
Ponto de vista da aplicação

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

Interação cliente-servidor: UDP



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();
```

Cria stream de entrada

Cria socket cliente

Translada nome do hospedeiro para endereço IP usando DNS

Exemplo: cliente Java (UDP)

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

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

      clientSocket.send(sendPacket);
```

Envia datagrama para servidor

```
      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)
        {
```

Cria espaço para datagramas recebidos

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

Recebe datagrama

```
        serverSocket.receive(receivePacket);
```

Exemplo: servidor Java (UDP)

```
                String sentence = new String(receivePacket.getData());
Obtém endereço IP
e número da porta   InetAddress IPAddress = receivePacket.getAddress();
do transmissor      int port = receivePacket.getPort();
                        String capitalizedSentence = sentence.toUpperCase();
                sendData = capitalizedSentence.getBytes();
Cria datagrama  DatagramPacket sendPacket =
para enviar ao cliente   new DatagramPacket(sendData, sendData.length, IPAddress,
                                port);
                serverSocket.send(sendPacket);
Escreve o       }
datagrama para  }
dentro do socket}
```

Termina o while loop, retorna e espera por outro datagrama

Programação de Sockets: referências

- Tutorial sobre Java: "Socket Programming in Java: a tutorial,"
  http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html
- Tarefas de programação
  http://wps.prenhall.com/wps/media/objects/9431/9657438/Tarefas_de_programacao/ProgrammingAssignment1.zip
- Referências:
  - http://wps.prenhall.com/wps/media/objects/9431/9657438/Conteudo_edicoes_ant/Building_a_simple_web_server.zip
  - http://wps.prenhall.com/br_kurose_redes_5/

Tarefas de programação de sockets

Tarefa 1: Servidor Web Multithread (ProgrammingAssignment1.zip)

http://wps.prenhall.com/br_kurose_redes_5/

Ao final desta tarefa de programação, você terá desenvolvido, em Java, um servidor Web multithread, que seja capaz de atender várias requisições em paralelo. Você implementará a versão 1.0 do HTTP como definida na RFC 1945.

Lembre-se de que o HTTP/1.0 cria uma conexão TCP separada para cada par requisição/resposta. Cada uma dessas conexões será manipulada por uma thread. Haverá também uma thread principal, no qual o servidor ficará à escuta de clientes que quiserem estabelecer conexões. Para simplificar o trabalho de programação, desenvolveremos a codificação em dois estágios. No primeiro estágio,você implementará um servidor multithread que simplesmente apresenta o conteúdo da mensagem de requisição HTTP que recebe. Depois que esse programa estiver executando normalmente, você adicionará a codificação necessária para gerar uma resposta apropriada.

Ao desenvolver a codificação, você poderá testar seu servidor com um browser Web. Mas lembre-se de que você não estará atendendo através da porta padrão 80, portanto, precisará especificar o número de porta dentro da URL que der a seu browser. Por exemplo, se o nome de seu hospedeiro for host.someschool.edu, seu servidor estiver à escuta na porta 6789 e você quiser obter o arquivo index.html, então deverá especificar a seguinte URL dentro do browser:

http://host.someschool.edu:6789/index.html

Quando seu servidor encontrar um erro, deverá enviar uma mensagem de resposta com uma fonte HTML adequada, de modo que a informação de erro seja apresentada na janela do browser.

Pode ser feito individual ou em dupla **Entrega até 26/09**