LABORATORIO DE PROGRAMACIÓN ORIENTADA A OBJETOS

DRA. MARICELA BRAVO

DIVISION DE CIENCIAS BASICAS E INGENIERIA
UAM - Azcapotzalco

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
Azcapotzalco

PROGRAMACIÓN CON SOCKETS

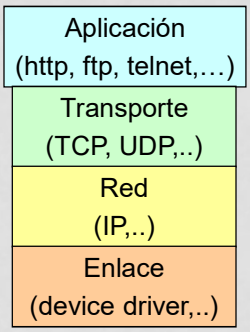
LABORATORIO DE PROGRAMACIÓN ORIENTADA A OBJETOS

2

PILA DE PROTOCOLOS TCP/IP

- **Capa de Aplicación**
  - Aplicaciones estándar
    - HTTP
    - FTP
    - Telnet
  - Aplicaciones de usuario
- **Capa de Transporte**
  - TCP
  - UDP
  - Interfaces de programación:
    - Sockets
- **Capa de Red**
  - IP
- **Capa de Enlace**
  - Drivers de dispositivos

- TCP/IP Stack

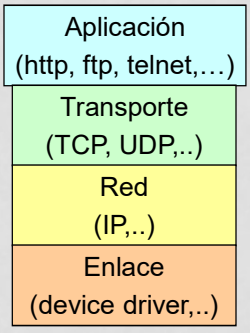


3

PILA DE PROTOCOLOS TCP/IP

- TCP (Transport Control Protocol)
- Es un protocolo orientado a conexión que proporciona un flujo de datos confiable entre dos computadoras.
- Ejemplo de aplicaciones:
  - HTTP
  - FTP
  - Telnet

- TCP/IP Stack

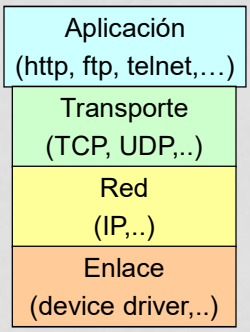


4

PILA DE PROTOCOLOS TCP/IP

- UDP (User Datagram Protocol)
- Es un protocolo que envía paquetes de datos independientes, llamados datagramas, de una computadora a otra, sin garantizar su llegada.
- Ejemplo de aplicaciones:
  - Clock server
  - Ping

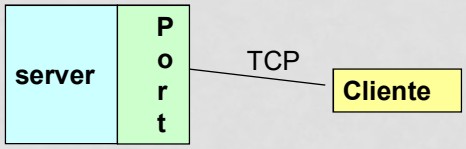
- TCP/IP Stack

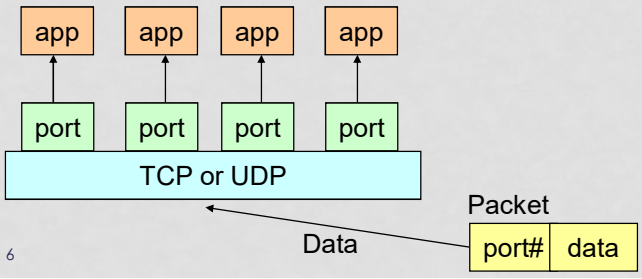


5

¿QUÉ SON LOS PUERTOS?

- TCP y UDP utilizan puertos para enviar datos entrantes a un proceso particular que se esté ejecutando en la computadora.





6

¿QUÉ SON LOS PUERTOS?

- Los puertos son representados por valores enteros positivos de 16 bits.
- Algunos puertos están reservados para soportar servicios preestablecidos:
  - FTP 21/TCP
  - Telnet 23/TCP
  - SMTP 25/TCP
  - HTTP 80/TCP
- Los procesos o servicios de usuarios generalmente usan números de puertos >= 1024.

7

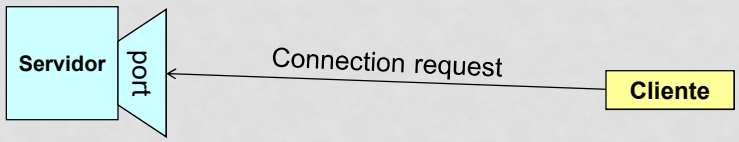
SOCKETS

- Los sockets proporcionan una interfaz para la programación de redes en la capa de transporte.
- Las comunicaciones de redes utilizando Sockets es muy similar al manejo de I/O en archivos.
  - El manejo de sockets es tratado como el manejo de archivos.
  - Los flujos utilizados en operaciones de I/O de archivos también son aplicables a I/O basado en sockets.
- La comunicación basada en Sockets es independiente del lenguaje de programación.
  - Esto es, que un programa de socket escrito en Java también se puede comunicar con un programa escrito en Java o con un programa de socket no escrito en Java.

8

COMUNICACIÓN ENTRE SOCKETS

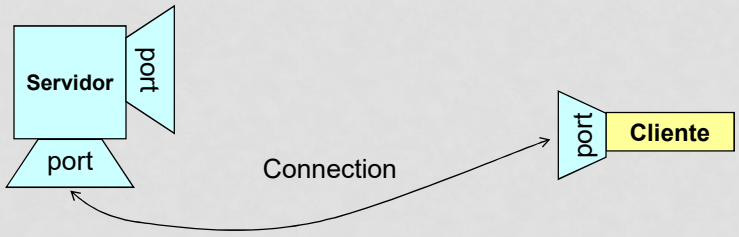
- Un servidor es un programa que se ejecuta en una computadora específica y tiene un socket que se asocia con un puerto específico.
- El servidor se mantiene en espera escuchando al socket para cuando un cliente realiza una petición de conexión.



9

COMUNICACIÓN ENTRE SOCKETS

- Si todo sale bien, el servidor acepta la conexión. Después de la aceptación, el servidor obtiene un nuevo socket asociado a un puerto diferente. Necesita un nuevo socket (consecuentemente un número de puerto diferente), de tal forma que puede continuar escuchando al socket original para solicitudes de conexión mientras que atiende al cliente conectado.



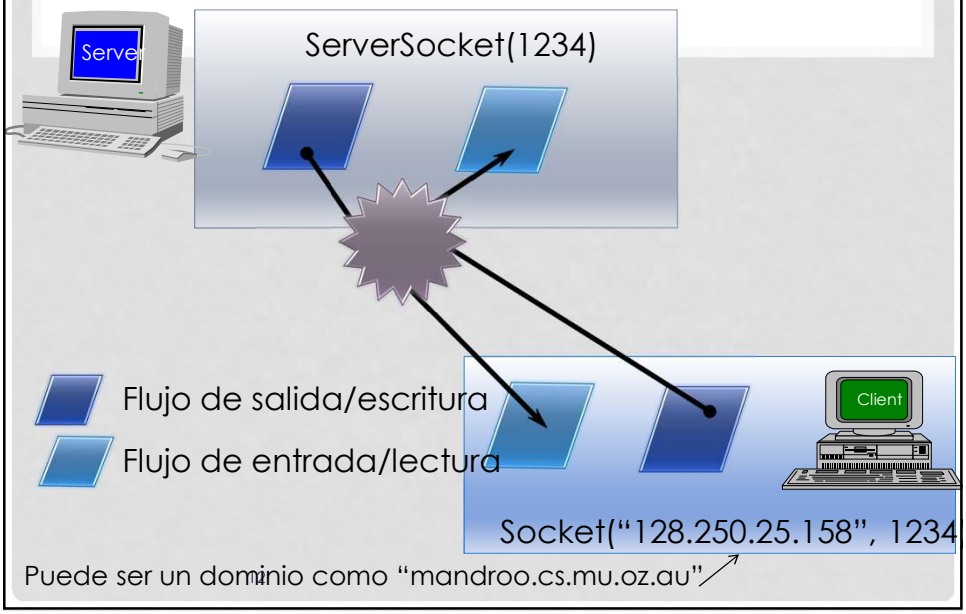
10

CLASES DE JAVA PARA CREAR SOCKETS

- Un socket es un endpoint de un enlace de comunicación bi-direccional entre dos programas ejecutándose en la red.
- Un socket se asocia a un número de puerto de tal forma que la capa de TCP puede identificar la aplicación a la cual están destinados los datos.
- El paquete de Java .net proporciona dos clases:
  - Socket – para implementar un cliente
  - ServerSocket – para implementar un servidor.

11

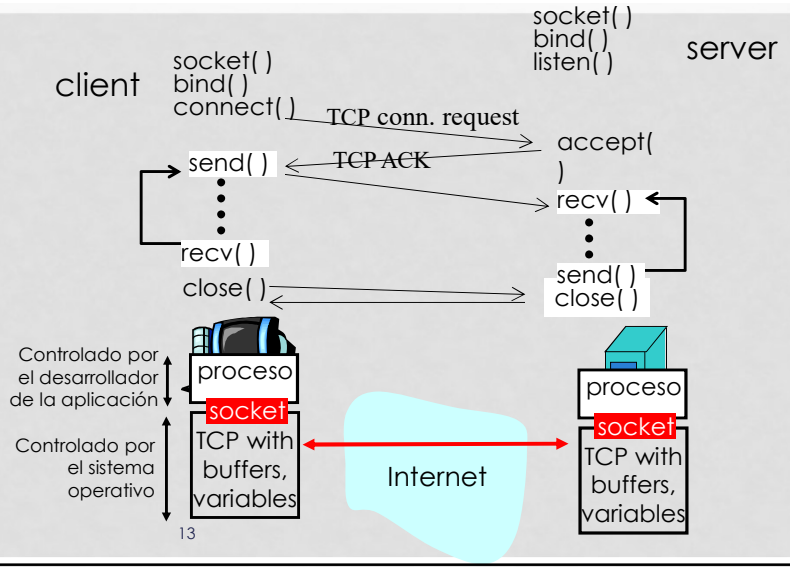
JAVA SOCKETS



12

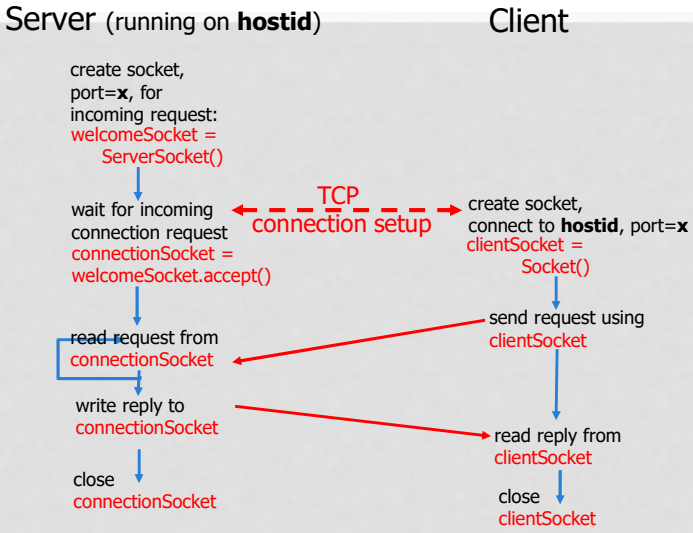
PROGRAMACIÓN DE SOCKETS USANDO TCP

TCP service: transferencia confiable de flujos de bytes



13

INTERACCIÓN DE SOCKETS TCP EN LA ARQUITECTURA CLIENTE/SERVIDOR



EJEMPLO DE UN SERVER

1. Crear el Server Socket:

```
ServerSocket server;
DataOutputStream os;
DataInputStream is;
server = new ServerSocket( PORT );
```

2. Espera solicitudes de clientes:

```
Socket client = server.accept();
```

3. Crea flujos de I/O para comunicarse con el cliente

```
is = new DataInputStream( client.getInputStream() );
os = new DataOutputStream( client.getOutputStream() );
```

4. Realiza comunicación con un cliente

```
Receive from client: String line = is.readLine();
Send to client: os.writeBytes("Hola\n");
```

5. Cierra el socket: `client.close();`

15

EJEMPLO DE UN CLIENTE

1. Crear un objeto de Socket:

```
cliente = new Socket( server, port_id );
```

2. Crea flujos de I/O para comunicarse con el servidor.

```
is = new DataInputStream(cliente.getInputStream() );
os = new DataOutputStream( cliente.getOutputStream() );
```

3. Realiza I/O o comunicación con el server:
   - Recibe datos del servidor:

     ```
     String line = is.readLine();
     ```

   - Envia datos al servidor:

     ```
     os.writeBytes("Hola\n");
     ```

4. Cierra el socket cuando termina:

```
cliente.close();
```

16

UN SERVER SIMPLE

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

   public class SimpleServer
   {
    public static void main(String args[]) throws IOException
    {
     // Registrar el servicio en el puerto 1234
     ServerSocket s = new ServerSocket(1245);
     //Espera y acepta conexiones
     Socket s1 = s.accept();
     //Obtiene un flujo de comunicación asociado con el socket
     OutputStream s1out = s1.getOutputStream();
     DataOutputStream dos = new DataOutputStream (s1out);
     //Envia un mensaje
     dos.writeUTF("Hola que tal");
     //Cierra la conexión, pero no el socket del servidor
     dos.close();
     s1out.close();
     s1.close();
    }

}
```

17

UN CLIENTE SIMPLE

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

public class SimpleClient
{
 public static void main(String args[]) throws IOException
 {
  //Abrir una conexión al server en el puerto 1234
  Socket s1 = new Socket("localhost",1245);
  //Obtener un manejador de flujo de entrada del socket y leer la entrada
  InputStream s1In = s1.getInputStream();
  DataInputStream dis = new DataInputStream(s1In);
  String st = new String (dis.readUTF());
  System.out.println(st);
  //Cerrar la conexion
  dis.close();
  s1In.close();
  s1.close();
 }
}
```

18

EJECUCIÓN

- Ejecutar Server en el localhost
  - java SimpleServer
- Ejecutar el Client en cualquier máquina:
  - java SimpleClient
    Hola que tal
- Si se ejecuta el cliente cuando el server no está escuchando:
  - java SimpleClient

```
Exception in thread "main" java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:320)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:133)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:120)
        at java.net.Socket.<init>(Socket.java:273)
        at java.net.Socket.<init>(Socket.java:100)
        at SimpleClient.main(SimpleClient.java:6)
```

19

SERVIDOR DE CHAT

20

```
public class ServidorChat
{
        private static int port = 1001;
        public static void main (String[] args) throws IOException
        {
                ServerSocket server = null;
                try{
                        server = new ServerSocket(port);
                 } catch (IOException e)
                {
                        System.err.println("No puede escuchar en el puerto: " + port);
                        System.err.println(e); System.exit(1);
                }
                Socket cliente = null;
                try{
                        cliente = server.accept();
                }catch (IOException e)
                {
                        System.err.println("Falló la aceptación de la conexión");
                        System.err.println(e);
                        System.exit(1);
                }
                BufferedReader in =
                new BufferedReader(new InputStreamReader( cliente.getInputStream()));
                String mensaje;
                while ((mensaje = in.readLine()) != null)
                {
                        System.out.println("El cliente dice: " + mensaje);
                } } }
```

21

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

public class ClienteChat
{
        private static int port = 1001;
        private static String host = "localhost";
        public static void main (String[] args) throws IOException
        {
                Socket servidor;
                PrintWriter out = null;
                try
                {
                        servidor = new Socket(host, port);
                        out = new PrintWriter(servidor.getOutputStream(), true);
                }
                catch (UnknownHostException e)
                {
                        System.err.println(e);
                        System.exit(1);
                }
                BufferedReader entrada =
                new BufferedReader( new InputStreamReader(System.in));
                String msg;
                while ((msg = entrada.readLine()) != null)
                {
                        out.println(msg);
                } } }
```

22

SERVIDOR DE ECO

SERVIDOR DE ECO

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

public class ServidordeEco
{
   public static void main(String[] args)
   {
          try
          {
            ServerSocket s = new ServerSocket(8189);
            Socket entrante = s.accept();
            try
            {
              InputStream se = entrante.getInputStream();
              DataInputStream in = new DataInputStream(se);

              OutputStream so = entrante.getOutputStream();
              PrintWriter out = new PrintWriter(so,true);
```

SERVIDOR DE ECO

```
            out.println("Escriba ADIOS para salir");
            boolean terminado = false;
            while(!terminado)
            {
            String linea = in.readLine();
            out.println("Eco:  " + linea );
            if (linea.trim().equals("ADIOS"))
                        terminado = true;
            }
      }finally
            {
                        entrante.close();
            }
      }
      catch(IOException e)
      {
            e.printStackTrace();
      }
      }
}
```

25

SERVIDOR DE WEB

26

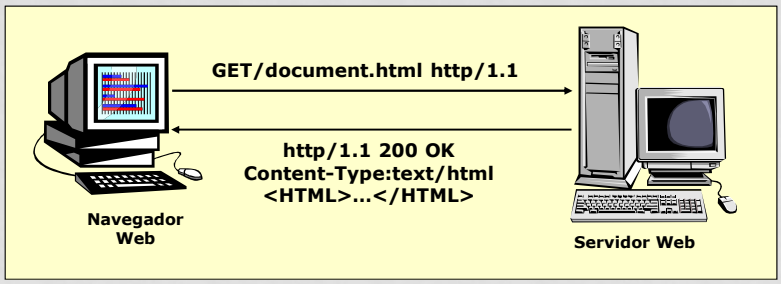
¿QUÉ ES UN SERVIDOR WEB?

- Un servidor Web HTTP es un programa que controla el flujo de datos entrantes y salientes de una computadora conectada a Intranet e Internet.
- Un servidor Web es un programa de aplicación que atiende las solicitudes HTTP realizadas por los navegadores.
- Escucha peticiones en el número de puerto 80, normalmente.

27

PROTOCOLO HTTP

- Es un protocolo de petición/respuesta sin estado cuya operación básica es la siguiente :



28

SERVIDOR WEB

- Maneja solamente una petición HTTP
- Acepta y parsea la petición HTTP
- Obtiene el archivo requerido del sistema de archivos del servidor
- Crea un mensaje de respuesta HTTP, el cual consiste del archivo precedido por líneas de cabecera.
- Envía la respuesta directamente al cliente.

29

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

public class WebServer
{

  public static void main(String[] args)
  {
     String requestMessageLine;
     String fileName;
     try
      {
        ServerSocket listenSocket = new ServerSocket(8000);
        Socket connectionSocket = listenSocket.accept();

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

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

        requestMessageLine = inFromClient.readLine();
        StringTokenizer tokenizedLine = new StringTokenizer(requestMessageLine);

                if (tokenizedLine.nextToken().equals("get"))
                {
                 fileName = tokenizedLine.nextToken();
                            if (fileName.startsWith("/") == true )
                              fileName = fileName.substring(1);
```

30

```
File file = new File(fileName);

int numOfBytes = (int) file.length();
FileInputStream inFile = new FileInputStream (fileName);
byte[] fileInBytes = new byte[numOfBytes];
inFile.read(fileInBytes);
outToClient.writeBytes("HTTP/1.1 200 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");
```

31

```
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");
}
catch (IOException e)
{
        e.printStackTrace();
}
}

}
```

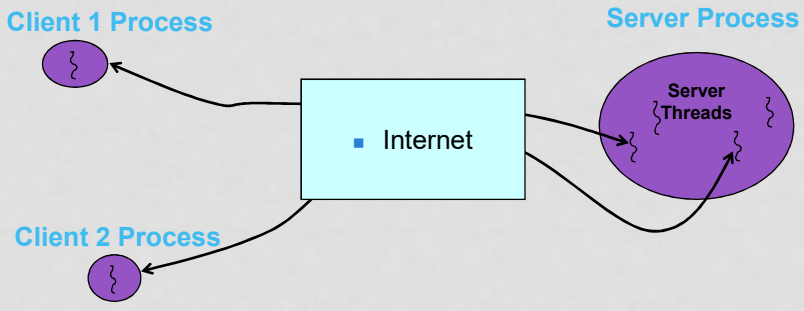
32

SERVIDOR EN UN CICLO: SIEMPRE EN EJECUCIÓN

```
import java.net.*;
import java.io.*;
public class SimpleServerLoop
{
  public static void main(String args[]) throws IOException
  {
    ServerSocket s = new ServerSocket(1234);
    while(true)
    {
          Socket s1=s.accept();
       OutputStream s1out = s1.getOutputStream();
          DataOutputStream dos = new DataOutputStream (s1out);
          dos.writeUTF("Hola");
          dos.close();
          s1out.close();
          s1.close();
    }
  }
}
```

33

SERVIDOR MULTI-HILOS: PARA SERVIR A MÚLTIPLES CLIENTES CONCURRENTEMENTE



34

SERVIDOR DE ECO CON HILOS

```
import java.io.*;
import java.net.*;
import java.util.*;
public class servidorEcoconHilos
{
        public static void main(String[] args) {
          try {
            int i = 1;
            ServerSocket s = new ServerSocket(8189);
            while(true)
             {
                  Socket entrante = s.accept();
                  System.out.println("generando hilo " + i);
                  Runnable r = new ManejadorHilos(entrante, i);
                  Thread t = new Thread(r);
                  t.start();
                  i++;
             }
           }
```

35

```
    catch(IOException e) {
                    e.printStackTrace();
              }
          }
}
//Clase para el manejo de hilos
Class ManejadorHilos implements Runnable
{
  public ManejadorHilos(Socket i, int c)
  {
    entrante = i;
    contador = c;
   }

  public void run()
  {
    try {
          try {
            InputStream secuenciaEntrada = entrante.getInputStream();
            OutputStream secuenciaSalida = entrante.getOutputStream();
             Scanner in = new Scanner(secuenciaEntrada);
             PrintWriter out = new PrintWriter(secuenciaSalida, true);
             out.println("Escriba ADIOS para salir");
```

36

```
//Reproducir la entrada del cliente
boolean terminado = false;
while(!terminado && in.hasNextLine()) {
   String linea = in.readLine();
   out.println("Eco:  " + linea );
   if (linea.trim().equals("ADIOS"))
                terminado = true;
}
}
finally
{
   entrante.close();
}
Catch(IOException e)
{
    e.printStackTrace();
}
}//End del run

private Socket entrante;
private int contador;
}
```

37

SOCKET EXCEPTIONS

```
try {
   Socket client = new Socket(host, port);
   handleConnection(client);
}
catch(UnknownHostException uhe) {
   System.out.println("Unknown host: " + host);
   uhe.printStackTrace();
}
catch(IOException ioe) {
System.out.println("IOException: " + ioe);
   ioe.printStackTrace();
}
```

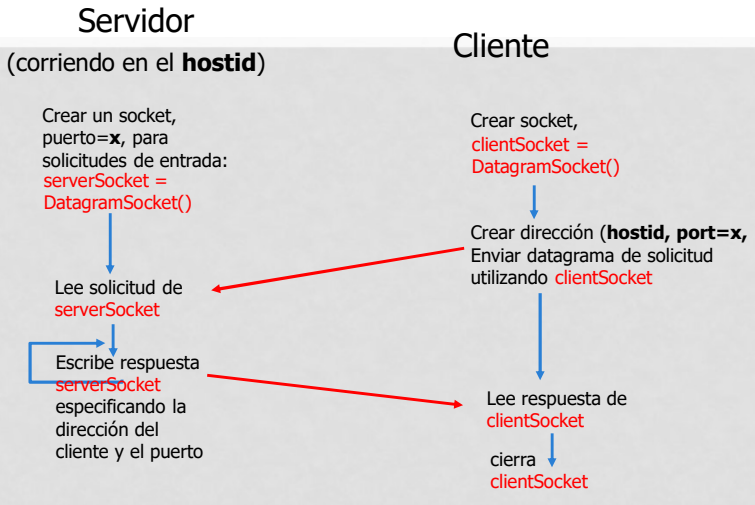
38

PROGRAMACIÓN DE SOCKETS CON UDP

- UDP
  - No es orientado a conexión y el servicio no es confiable.
  - No existe una fase inicial de handshaking.
  - Los datos transmitidos pueden recibirse fuera de orden.
- Programación de Sockets con UDP
  - No necesita que se asocien flujos al socket.
  - El host que envía crea paquetes adjuntando la dirección IP destino y el número de puerto en cada lote de bytes.
  - El proceso que recibe debe reorganizar el paquete para obtener los bytes de información del paquete.

39

INTERACCIÓN CLIENTE/SERVIDOR CON SOCKETS UDP



40

JAVA UDP SOCKETS

- Paquete java.net
  - java.net.DatagramSocket
    - Un socket para enviar y recibir paquetes de datagramas.
    - Constructores y métodos
      - DatagramSocket(int port): Construye un socket de datagrama y lo asocia con un puerto espefífico en la máquina local.
      - void receive(DatagramPacket p): Recibe paquetes.
      - void send(DatagramPacket p): Envía paquetes.
      - void close()

41

UDPCLIENT.JAVA

```
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();
```

42

UDPCLIENT.JAVA

```
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();

   }
}
```

43

UDPSERVER.JAVA

```
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);

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

44

UDPSERVER.JAVA

```
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);

    }
  }
}
```

45

DUDAS O COMENTARIOS???