Redes de computadoras

Programación de sockets

Las diapositivas están basadas en en libro:

“Redes de Computadoras – Un enfoque descendente”
de James F. Kurose & Keith W. Ross

Aplicaciones de red

Distintos tipos de aplicaciones de red

Implementar un protocolo RFC

- Los programas deben seguir las reglas dispuestas en el RFC
- Programadores independientes y sin ningún tipo de contacto pueden trabajar en un programa cliente y servidor que luego se conecten correctamente.

Implementar un protocolo propietario

- No requieren seguir reglas establecidas en un RFC.
- El equipo de desarrollo debe coordinar el funcionamiento de los programas servidor y cliente. Deben tener la precaución de no utilizar puertos ya reservados por una especificación RFC.

Aplicaciones de red

Implementación de un Protocolo propietario

¿Sobre TCP o UDP?

TCP

- Orientado a conexión
- Proporciona un canal fiable de flujo

UDP

- Transporta paquetes independientes de un sistema terminal a otro
- Sin ninguna garatía acerca de la entrega

Aplicaciones de red

Por lo general las aplicaciones de red constan en algún punto de un programa cliente y un programa servidor.

Los procesos de estos programas se comunican entre sí escribiendo y leyendo en sockets.

Al momento de navegar por la WWW se utilizará un navegador que funcionará como cliente. Al solicitar una página se creará un socket que quedará asociado a esa solicitud y será el punto de conexión con el servidor, el cual estará escuchando por un socket asociado a un puerto y en algunos casos creará un socket para responder la solicitud.

Sockets

Los sockets son una interfaz que permite la comunicación entre procesos en un mismo equipo así como entre procesos en equipos diferentes.

Están asociados a una dirección y un puerto en el host.

Sockets

Tipos de sockets

Sockets de Flujo

Vía TCP, el cliente solicita la conexión, el servidor acepta o rechaza y en caso de ser aceptada se da la transferencia de datos.

Sockets de Datagramas

Utiliza UDP, (User Datagram Protocol) se crea un paquete de datos con la información de su destino y se envía sin establecer una previa conexión.

Sockets de flujo - TCP

- **s1** El servidor debe estar activo y debe haber creado un socket el cual esperará el contacto del cliente en una IP y un puerto.

- **c1** El cliente crea un socket local e inicia un contacto con el servidor. (Acuerdo en tres fases, three way handshake)

- **s2** Cuando es contactado por el cliente el servidor crea un nuevo socket a través del cual el proceso servidor se comunicará con el cliente.

  - Permite que un servidor dialogue con muchos clientes
  - Sólo puede haber un socket asociado a un puerto

Sockets de flujo - TCP

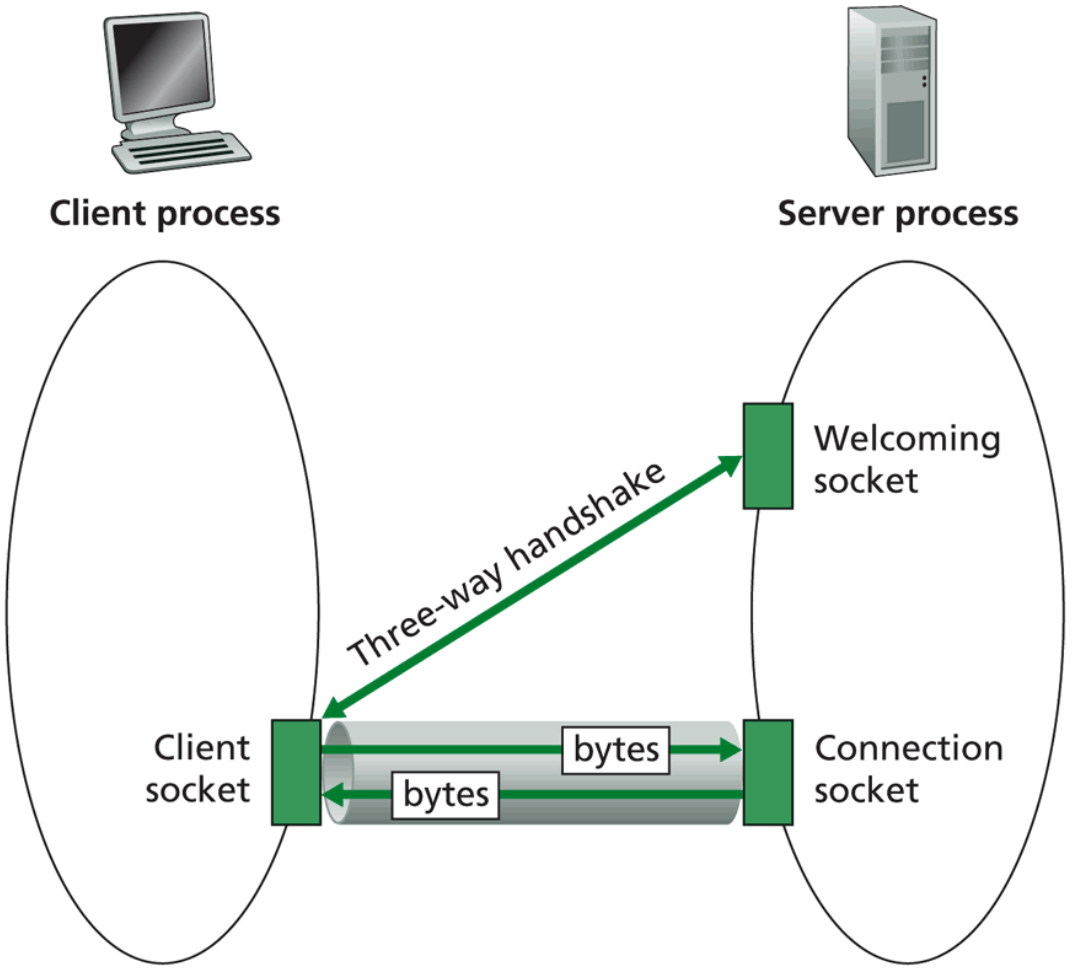

Figure 2.27 ♦ Client socket, welcoming socket, and connection socket

Ejemplo de aplicación cliente-servidor en Java

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

class ClienteTCP {

    public static void main(String argv[]) throws Exception {
        String frase;
        String fraseModificada;

        BufferReader entradaDesdeUsuario = new BUfferedReader(
            new InputStreamReader(System.in)
        );

        Socket socketCliente = new Socket("nombrehost", 6789);
        DataOutputStream salidaAServidor = new DataOutputStream(
            socketCliente.getOutputStream()
        );

        BufferedReader entradaDesdeServidor = new BufferedReader(
            new InputStreamReader(socketCliente.getInputStream())
        );

        frase = entradaDesdeUsuario.readLine();
        salidaAServidor.writeBytes(frase + '\n');

        fraseModificada = entradaDesdeServidor.readLine();

        System.out.println("DEL SERVIDOR: " + fraseModificada);
        socketCliente.close();
    }

}
```

Aplicación cliente servidor sencilla donde el cliente envía un texto al servidor y el servidor lo retorna en mayúsculas.

Cliente TCP - Importaciones

java.io

El paquete contiene clases para los flujos de entrada y de salida.

BufferedReader y DataOutputStream

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

Java.net

Contiene clases para el soporte de red.

Socket y ServerSocket

Cliente TCP – flujo de entrada

Se crea un objeto de flujo “entradaDesdeUsuario” del tipo BufferedReader.

El flujo se inicializa con System.in lo que indica que el flujo se asociará a la entrada estándar.

```
10          BufferReader entradaDesdeUsuario = new BUfferedReader(
11              new InputStreamReader(System.in)
12          );
```

Cliente TCP – socket cliente

```
14          Socket socketCliente = new Socket("nombrehost", 6789);
```

El objeto socketCliente de tipo Socket inicia la conexión TCP entre el cliente y el servidor.

El objeto Socket se inicializa con el nombre de host del servidor y un número de puerto .

El mismo número de puerto utilizado por el cliente en esta línea debe ser por el cual el servidor estará “escuchando”.

La dirección IP del servidor junto con el puerto identifican a un proceso.

Cliente TCP – flujos asociados al socket

```
16          DataOutputStream salidaAServidor = new DataOutputStream(
17              socketCliente.getOutputStream()
18          );
19
20          BufferedReader entradaDesdeServidor = new BufferedReader(
21              new InputStreamReader(socketCliente.getInputStream())
22          );
```

Se crean los flujos asociados al socket.

SalidaAServidor proporciona la salida del proceso hacia el socket.

EntradaDesdeServidor la entrada a partir del socket.

Cliente TCP – envío

```
24          frase = entradaDesdeUsuario.readLine();
```

Se captura el texto ingresado por el usuario en la variable “frase” hasta que el usuario introduce un retorno de carro.

```
25          salidaAServidor.writeBytes(frase + '\n');
```

Se introduce el texto en el flujo salidaAServidor.

Esta cadena fluye a través del socket del cliente y entra en el canal TCP.

El cliente espera entonces a recibir la respuesta del servidor.

Cliente TCP – recepción del mensaje

```
27          fraseModificada = entradaDesdeServidor.readLine();
```

Cuando llegan caracteres desde el servidor, entran en el flujo entrada dDesdeServidor y se almacenan en la cadena fraseModificada.

Los caracteres continúan acumulándose en fraseModificada hasta que la línea termina con un carácter de retorno de carro.

Cliente TCP – recepción del mensaje

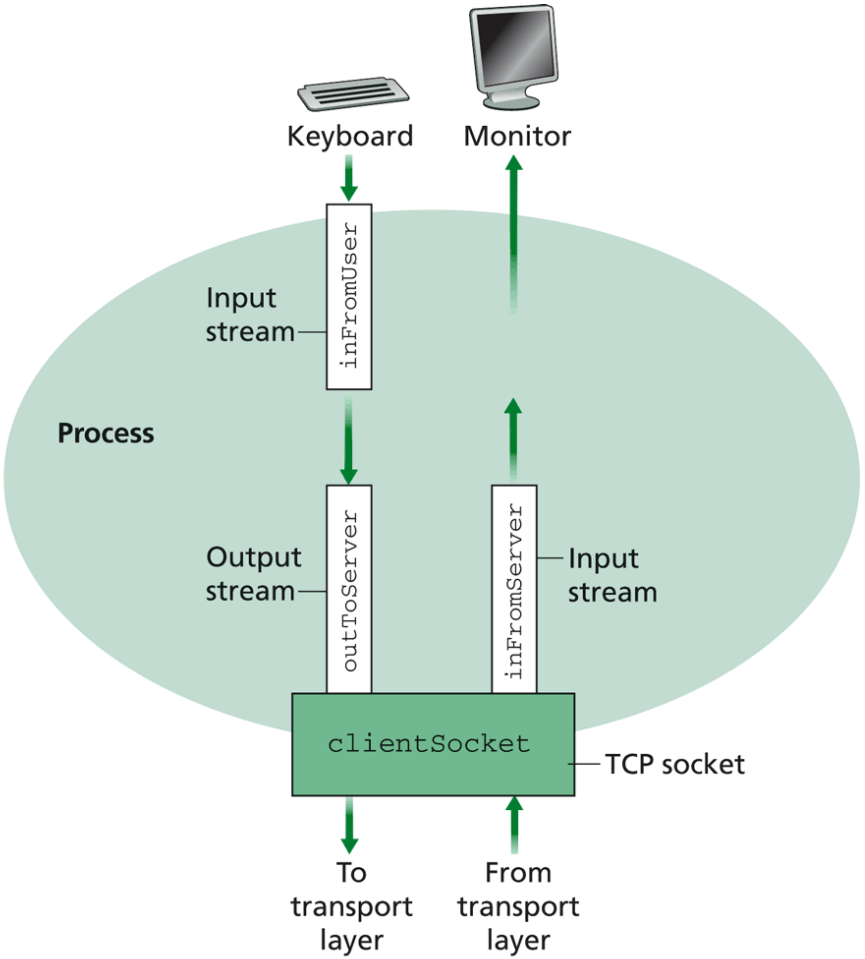

Figure 2.29 ♦ `TCPClient` has three streams through which characters flow.

Cliente TCP – cierre de la conexión

Se imprime en la salida estándar la cadena devuelta por el servidor.

```
29          System.out.println("DEL SERVIDOR: " + fraseModificada);
30          socketCliente.close();
```

Se cierra el socket, y así también la conexión TCP entre cliente y servidor.

Este acto hace que se envíe desde el cliente
un mensaje TCP a TCP al servidor.

Servidor TCP

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

/**
 * ServidorTCP
 */
public class ServidorTCP {

    public static void main(String[] args) throws Exception {
        String fraseCliente;
        String fraseMayusculas;

        ServerSocket socketBienvenida = new ServerSocket(6789);
        boolean cerrarConexion = false;

        while (!cerrarConexion) {
            Socket socketConexion = socketBienvenida.accept();

            BufferedReader entradaDesdeCliente = new BufferedReader(
                new InputStreamReader(
                    socketConexion.getInputStream()
                )
            );

            DataOutputStream salidaACliente = new DataOutputStream(
                socketConexion.getOutputStream()
            );

            fraseCliente = entradaDesdeCliente.readLine();

            fraseMayusculas = fraseCliente.toUpperCase() + '\n';
            salidaACliente.writeBytes(fraseMayusculas);

            if (fraseMayusculas == "EXIT") {
                cerrarConexion = true;
            }
        }

        socketBienvenida.close();
    }

}
```

Servidor TCP – Socket de recepción

El primer punto de diferencia es el socket de bienvenida

```
13          ServerSocket socketBienvenida = new ServerSocket(6789);
```

Se crea un ServerSocket el cual estará “escuchando” a la espera de que un cliente inicie una conexión.

El parametro ingresado corresponde al número de puerto utilizado.

Servidor TCP – Socket de conexión

Cuando un cliente envía una solicitud el socket de bienvenida TCP crea un canal virtual directo entre “socketCliente” y “socketConexion”.

```
17            Socket socketConexion = socketBienvenida.accept();
```

El cliente y el servidor pueden entonces enviarse bytes a través del canal y todos los bytes llegan al otro lado en orden.

(Esta versión del programa no queda escuchando otras solicitudes al tiempo que responde una, esto se podría solucionar utilizando hilos)

Programación de sockets en TCP

- La comunicación se realiza como si existiera un conducto (pipe), el cual se mantiene hasta que uno de los procesos lo cierra.

- Los procesos envían bytes en el conducto sin asociar dirección de destino, el conducto está conectado lógicamente al destino.

- El conducto proporciona un canal fiable de datos, la secuencia de bytes recibida es exactamente igual a la emitida.

Programación de sockets en TCP

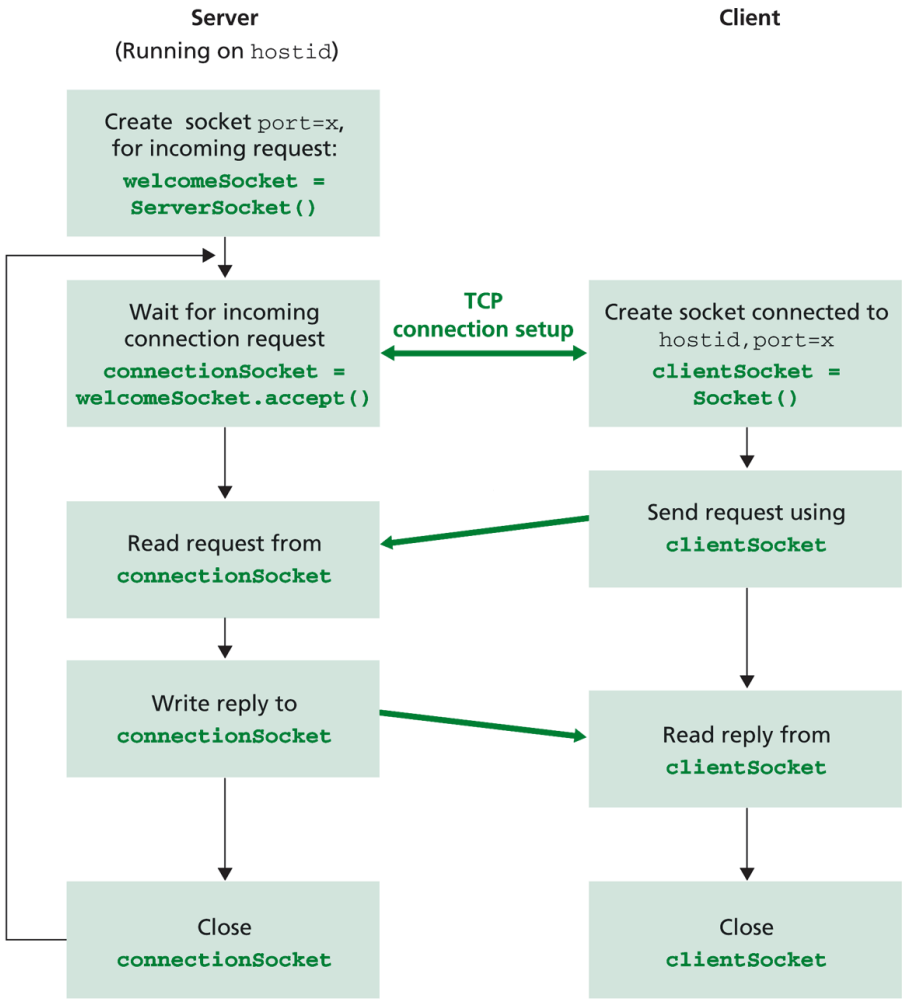

Figure 2.28 ♦ The client-server application, using connection-oriented transport services

Programación de sockets en UDP

- UDP es un servicio sin conexión.
- No existe la fase de negociación que proporciona el conducto.

  ¿socket de bienvenida?
- Se debe asociar la dirección de destino a cada lote de bytes que se desea enviar.
- lotes de bytes, dirección IP y puerto. Paquete.
- Servicio orientado al mensaje.
- Servicio de entrega de “best effort” o “fire and forget”.

Programación de sockets en UDP

- Se crea el paquete
- El proceso emisor lo pone en la red a través de un socket.
- No se garantiza que el paquete llegue al destino.

Cliente UDP

```java
public class ClienteUDP {

    public static void main(String[] args) throws Exception {
        BufferedReader entradaDesdeUsuario = new BufferedReader(
            new InputStreamReader(System.in)
        );

        DatagramSocket socketCliente = new DatagramSocket();
        InetAddress direccionIP = InetAddress.getByName("nombreDeHost");

        byte[] enviarDatos = new byte[1024];
        byte[] recibirDatos = new byte[1024];

        String frase = entradaDesdeUsuario.readLine();
        enviarDatos = frase.getBytes();

        DatagramPacket enviarPaquete = new DatagramPacket(
            enviarDatos, enviarDatos.length, direccionIP, 9876);

        socketCliente.send(enviarPaquete);

        DatagramPacket recibirPaquete =
            new DatagramPacket(recibirDatos, recibirDatos.length);

        socketCliente.receive(recibirPaquete);

        String fraseModificada = new String(recibirPaquete.getData());

        System.out.println("DEL SERVIDOR: " + fraseModificada);

        socketCliente.close();
```

Cliente UDP – constructor del socket

```
16        DatagramSocket socketCliente = new DatagramSocket();
```

Se crea el objeto socketCliente de tipo DatagramSocket.

A diferencia del cliente TCP en esta línea no se genera una conexión con el servidor.

Cliente UDP – búsqueda DNS

```
18          InetAddress direccionIP = InetAddress.getByName("localhost");
```

Para la comunicación con el host destino es necesario conocer su IP. En este caso se realiza una búsqueda DNS.

La versión TCP también invoca una búsqueda DNS, sólo que de manera implícita en lugar de explícita.

Cliente UDP – manejo de datos

Para los datos a enviar y recibir se utilizaran arreglos de bytes.

```
20          byte[] enviarDatos = new byte[1024];
21          byte[] recibirDatos = new byte[1024];
22
23          String frase = entradaDesdeUsuario.readLine();
24          enviarDatos = frase.getBytes();
```

Desde la entrada estándar se lee lo que ingresa el usuario en frase, y desde ahí se “castea” en un array de bytes.

Cliente UDP – Creación del paquete

Se crea el paquete que se introducirá en la red a través del socket.

```
26          DatagramPacket enviarPaquete = new DatagramPacket(
27              enviarDatos, enviarDatos.length, direccionIP, 9876);
```

El paquete incluye:

- los datos
- longitud de los datos
- dirección IP del destino
- puerto de destino

Cliente UDP – Envio y recepción

El objeto socketCliente envía los datos a la red.

```
29          socketCliente.send(enviarPaquete);
```

TCP insertaba la cadena de caracteres en un flujo de datos con una conexión lógica con el servidor,

UDP crea un paquete que incluye la dirección del servidor.

```
31          DatagramPacket recibirPaquete =
32              new DatagramPacket(recibirDatos, recibirDatos.length);
33          socketCliente.receive(recibirPaquete);
```

Se crea un recipiente para recibir el paquete y se bloquea hasta recibirlo.

Cliente UDP

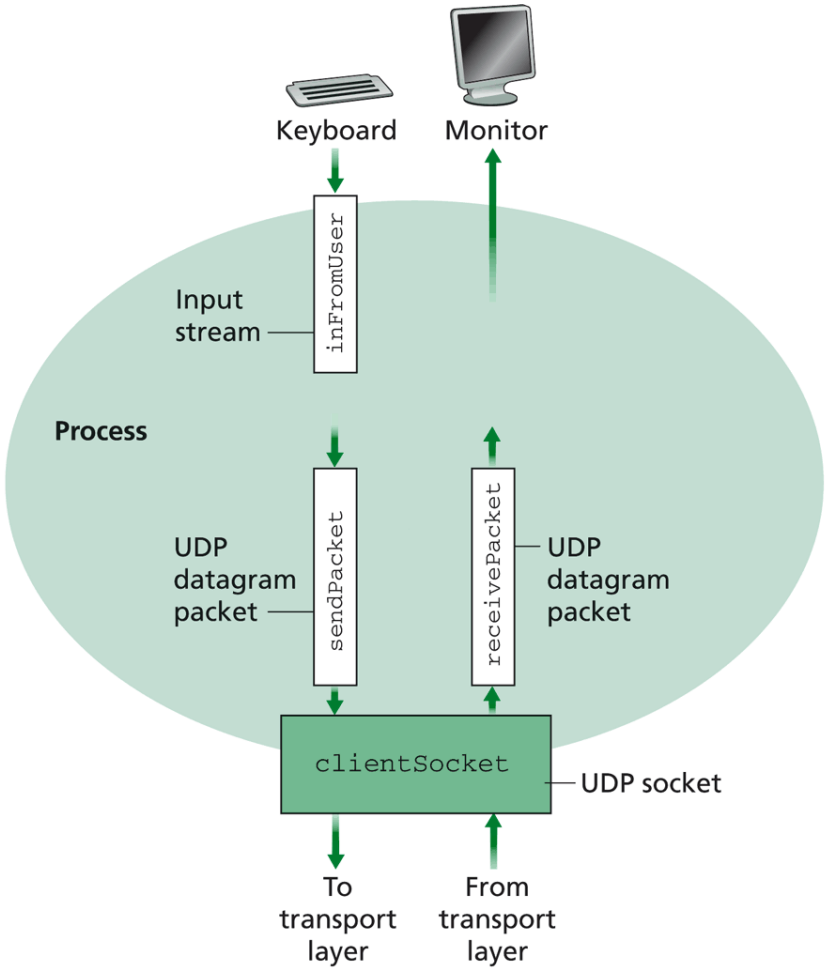

Figure 2.31 ♦ `UDPClient` has one stream; the socket accepts packets from the process and delivers packets to the process.

Servidor UDP

```java
public static void main(String[] args) throws Exception {
    DatagramSocket socketServidor = new DatagramSocket(9876);
    boolean ejecutarPrograma = true;

    byte[] recibirDatos = new byte[1024];
    byte[] enviarDatos = new byte[1024];

    while (ejecutarPrograma) {
        DatagramPacket recibirPaquete =
            new DatagramPacket(recibirDatos, recibirDatos.length);
        socketServidor.receive(recibirPaquete);

        String frase = new String(recibirPaquete.getData());
        InetAddress direccionIP = recibirPaquete.getAddress();
        int puerto = recibirPaquete.getPort();

        String fraseMayusculas = frase.toUpperCase();
        enviarDatos = fraseMayusculas.getBytes();

        DatagramPacket enviarPaquete =
            new DatagramPacket(enviarDatos,
                enviarDatos.length,
                direccionIP,
                puerto);

        socketServidor.send(enviarPaquete);
    }
```

Servidor UDP

Se crea un DatagramSocket al igual que en el lado del cliente, excepto por que se le asigna un número de puerto.

```
DatagramSocket socketServidor = new DatagramSocket(9876);
```

Todos los datos enviados y recibidos pasaran por este socket.

Ya que UDP proporciona un servicio sin conexión no será necesario crear otro socket para que siga escuchando.

Servidor UDP

Se decodifica el paquete

Se extrae la dirección IP y el puerto del cliente.

A diferencia de TCP no utilizan el mismo puerto

```
DatagramPacket recibirPaquete =
    new DatagramPacket(recibirDatos, recibirDatos.length);
socketServidor.receive(recibirPaquete);

String frase = new String(recibirPaquete.getData());
InetAddress direccionIP = recibirPaquete.getAddress();
int puerto = recibirPaquete.getPort();

String fraseMayusculas = frase.toUpperCase();
enviarDatos = fraseMayusculas.getBytes();

DatagramPacket enviarPaquete =
    new DatagramPacket(enviarDatos,
        enviarDatos.length,
        direccionIP,
        puerto);

socketServidor.send(enviarPaquete);
```

Servidor UDP

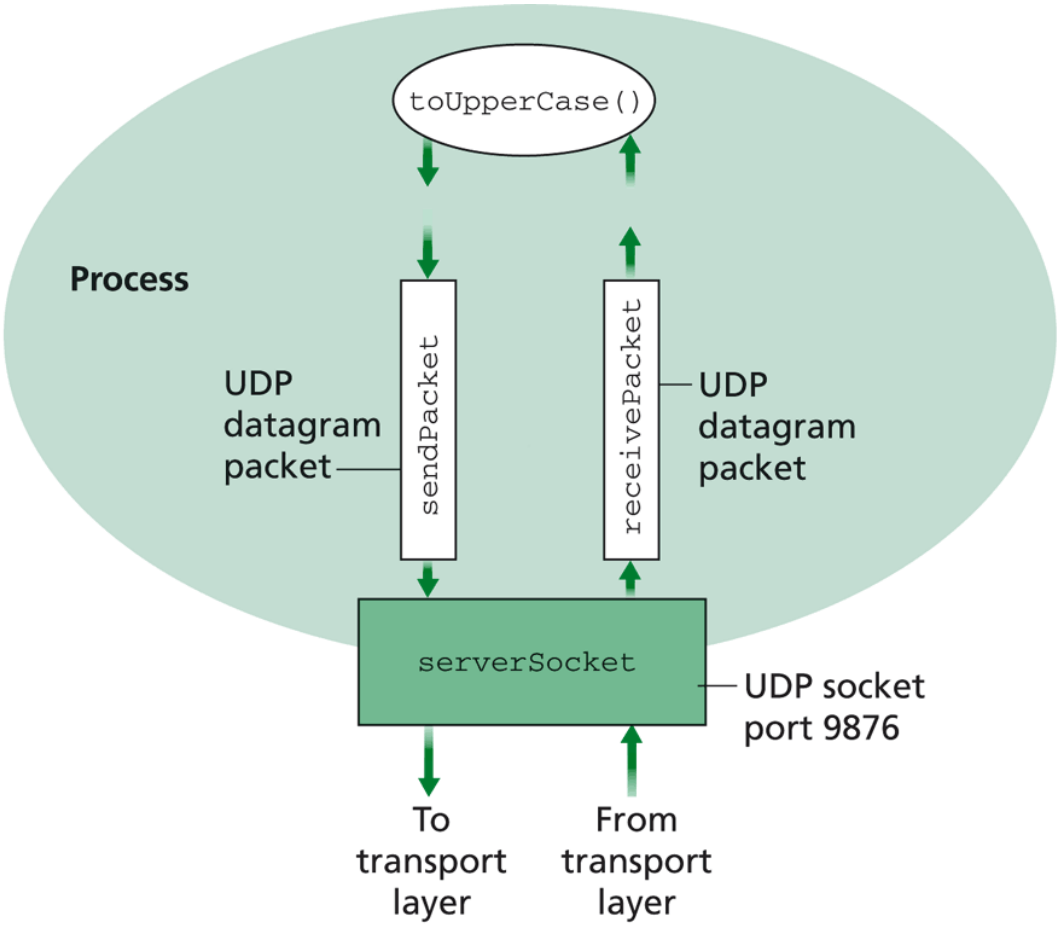

Figure 2.32 ♦ `UDPServer` has no streams; the socket accepts packets from the process and delivers packets to the process.

UDP

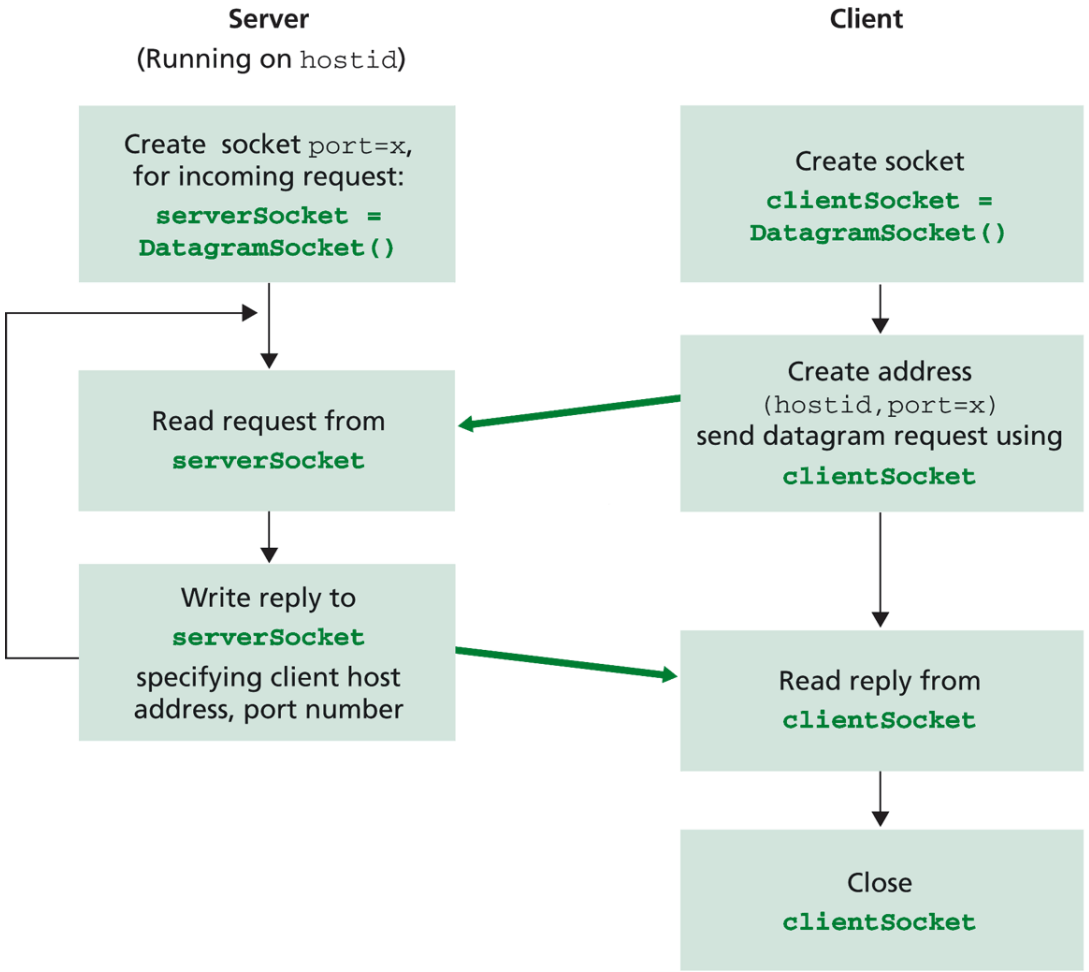

Figure 2.30 ♦ The client-server application, using connectionless transport services

Enlaces de interés

Diferencia entre Socket y puerto
https://stackoverflow.com/questions/152457/what-is-the-difference-between-a-port-and-a-socket

Windows API
https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516#user_input_and_messaging

Programación de sockets en C / C++

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Sockets

http://www.linuxhowtos.org/C_C++/socket.htm

http://www.bogotobogo.com/cplusplus/sockets_server_client.php

https://www.geeksforgeeks.org/socket-programming-cc/