	UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA **FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)** **DOCUMENTO DEL SISTEMA DE CALIDAD**

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC y LSC	2003-1	5038	Programación Orientada a Objetos 2

PRÁCTICA No.	LABORATORIO DE		DURACIÓN (HORA)
5	NOMBRE DE LA PRÁCTICA	Datagramas	4

1. INTRODUCCIÓN

En esta práctica se desarrollará un programa en java para comunicación entre procesos no orientada a conexión. El alumno establecerá un criterio comparativo sobre las diferencias que hay entre la comunicación TCP y UDP.

2. OBJETIVO (COMPETENCIA)

El alumno será capaz de desarrollar programas utilizando la comunicación entre procesos aplicando el protocolo UDP no orientado a conexión.

3. FUNDAMENTO

UDP (*User Datagram Protocol*) es un protocolo no orientado a la conexión, a diferencia de lo que ocurría en el protocolo TCP, en el cual si un paquete se dañaba durante la transmisión se reenviaba ese paquete, para asegurar una comunicación segura entre cliente y servidor; con UDP no hay garantía alguna de que los paquetes lleguen en el orden correcto a su destino y, ni tan siquiera hay seguridad de que lleguen todos los paquetes que se hayan enviado. La ventaja con UDP se refleja en la comunicación, ya que es más rápida (porque no hay que establecer conexiones ni circuitos virtuales entre máquinas).

La clase **DatagramPacket**, junto con la clase **DatagramSocket**, son las que se utilizan para la implementación del protocolo UDP. Para enviar datos a través de UDP, hay que construir un objeto de tipo **DatagramPacket** y enviarlo a través de un objeto **DatagramSocket**, y al revés para recibirlos, es decir, a través de un objeto **DatagramSocket** se recoge el objeto **DatagramPacket**. Toda la información respecto a la dirección, puerto y datos, está contenida en el paquete. El tamaño físico máximo de un datagrama es 65535 bytes, y teniendo en cuenta que hay que incluir datos de cabecera, esa longitud nunca está disponible para datos de usuario, sino que siempre es algo menor.

Clase DatagramPacket

Dispone de dos constructores; para enviar paquetes y recibir paquetes. Ambos requieren que se les proporcione un array de bytes y la longitud que tiene. En el caso de la recepción de datos, no es necesario nada más, los datos que se reciban se depositarán en el array; aunque, en el caso de que se reciban más datos físicos de los que soporta el array, el exceso de información se perderá y se lanzará una excepción de tipo IllegalArgumentException.

1. Constructores para datagramas que van a ser enviados: **DatagramPacket**(byte[] buf, int length) y **DatagramPacket**(byte[] buf, int length, InetAddress address, int port). Estos constructores crean una instancia de datagrama compuesta por una cadena de bytes que almacena el mensaje, la longitud del mensaje y la dirección de Internet y el número de puerto local del conector destino, tal y como sigue.

array de bytes que contiene el mensaje	longitud del mensaje	dirección Intenet	número de puerto

2. Constructores para datagramas recibido: **DatagramPacket**(byte[] buf, int offset, int length) y **DatagramPacket**(byte[] buf, int offset, int length, InetAddress address, int port). Estos constructores nos permiten crear instancias de los datagramas recibidos, especificando la cadena de bytes en la que alojar el mensaje, la longitud de la misma y el offset dentro de la cadena.

Dentro de esta clase hay métodos para obtener los diferentes componentes de un datagrama, tanto recibido como enviado:

a) getData() : Obtiene el mensaje contenido en el datagrama.
b) getAddress() : Obtiene la dirección IP.
c) getPort(): Obtiene el puerto.
d) getLength()**:** Devuelve el número de bytes que contiene la parte de datos del datagrama.

Clase DatagramSocket

Un objeto de la clase **DatagramSocket** puede utilizarse tanto para enviar como para recibir un datagrama. Proporciona tres constructores:

1. DatagramSocket(): constructor sin argumentos que permite que el sistema elija un puerto entre los que estén libres y selecciona una de las direcciones locales.
2. DatagramSocket(int port): constructor que toma un número de puerto como argumento, apropiado para los procesos que necesitan un número de puerto (servicios).
3. DatagramSocket(int port, InetAddress laddr): constructor que toma como argumentos el número de puerto y una determinada dirección local.

La clase DatagramSocket proporciona varios métodos entre ellos se encuentran los siguientes:

a) send(DatagramPacket p) y receive(DatagramPacket p): estos métodos sirven para transmitir datagramas entre un par de conectores. El argumento de *send* es una instancia de DatagramPacket conteniendo el mensaje y el destino. El argumento de *receive* es un DatagramPacket vacío en el que colocar el mensaje, su longitud y su origen. Ambos métodos pueden lanzar excepciones IOException.
b) setSoTimeout(int timeout): este método permite establecer un tiempo de espera límite. Cuando se fija un límite, el método *receive* se bloquea durante el tiempo fijado y después lanza una excepción InterruptedIOException.
c) connect(InetAddress address, int port): este método se utiliza para conectarse a un puerto remoto y a una dirección Internet concretos, en cuyo caso el conector sólo podrá enviar y recibir mensajes de esa dirección.

La forma genérica de programar con datagramas sería la siguiente:

1. **Servidor**
   a) Crear un socket para el datagrama en un puerto “conocido”.
   b) Crear un paquete de datagrama vació.
   c) Ligar el datagrama al socket y esperar los mensajes del cliente.
   d) Obtener los datos, dirección y puerto del cliente y mandarle una respuesta.

2. **Cliente**
   a) Crear un socket de datagrama en cualquier puerto disponible.
   b) Crear un paquete de datagrama con la dirección y puerto destino.
   c) Mandar el datagrama a través del socket.
   d) Crear un paquete vació reservado para la respuesta del servidor.
   e) Ligar el paquete al socket y esperar por la respuesta del servidor.
   f) Obtener los datos del paquete de datagrama y procesarlos.

A continuación se muestra un ejemplo sencillo en el que utilizamos ambas clases: una aplicación cliente/servidor en el que el servidor reenvía el mismo mensaje que le envían los diferentes clientes.

Código del Cliente

```
import java.net.*;

import java.io.*;

public class ClienteUDP {

  // Los argumentos proporcionan el mensaje y el nombre del servidor
  public static void main(String args[]) {

    try {
      DatagramSocket socketUDP = new DatagramSocket();
      byte[] mensaje = args[0].getBytes();
      InetAddress hostServidor = InetAddress.getByName(args[1]);
      int puertoServidor = 6789;

      // Construimos un datagrama para enviar el mensaje al servidor
      DatagramPacket peticion =
        new DatagramPacket(mensaje, args[0].length(), hostServidor,
                           puertoServidor);

      // Enviamos el datagrama
      socketUDP.send(peticion);

      // Construimos el DatagramPacket que contendrá la respuesta
      byte[] bufer = new byte[1000];
      DatagramPacket respuesta =
        new DatagramPacket(bufer, bufer.length);
```

```
      socketUDP.receive(respuesta);

      // Enviamos la respuesta del servidor a la salida estandar
      System.out.println("Respuesta: " + new String(respuesta.getData()));

      // Cerramos el socket
      socketUDP.close();

    } catch (SocketException e) {
      System.out.println("Socket: " + e.getMessage());
    } catch (IOException e) {
      System.out.println("IO: " + e.getMessage());
    }
  }
}
```

Código del Servidor

```
import java.net.*;

import java.io.*;

public class ServidorUDP {

  public static void main (String args[]) {

    try {

      DatagramSocket socketUDP = new DatagramSocket(6789);
      byte[] bufer = new byte[1000];

      while (true) {
        // Construimos el DatagramPacket para recibir peticiones
        DatagramPacket peticion =
          new DatagramPacket(bufer, bufer.length);

        // Leemos una petición del DatagramSocket
        socketUDP.receive(peticion);

        System.out.print("Datagrama recibido del host: " +
                         peticion.getAddress());
        System.out.println(" desde el puerto remoto: " +
                         peticion.getPort());

        // Construimos el DatagramPacket para enviar la respuesta
        DatagramPacket respuesta =
          new DatagramPacket(peticion.getData(), peticion.getLength(),
                           peticion.getAddress(), peticion.getPort());
```

```
        // Enviamos la respuesta, que es un eco
        socketUDP.send(respuesta);
      }

    } catch (SocketException e) {
      System.out.println("Socket: " + e.getMessage());
    } catch (IOException e) {
      System.out.println("IO: " + e.getMessage());
    }
  }

}
```

4 PROCEDIMIENTO (DESCRIPCIÓN)

A) EQUIPO NECESARIO	MATERIAL DE APOYO
Computadora con sistema operativo Linux	Práctica impresa

B) DESARROLLO DE LA PRÁCTICA

Realizar un programa el cual será un chat al cual se podrán manejar n clientes. El programa cliente enviará mensajes al programa servidor, y el programa servidor se encargara de reenviarlos a todos los clientes que se encuentren conectados.

El programa cliente será grafico, de tal forma que muestre en parte superior los mensajes que envían los clientes, y en la parte inferior será el área para que el usuario escriba el mensaje que quiere enviar al servidor y que será reenviado a todos los clientes.

Al conectarse el cliente al servidor, el cliente recibirá mensaje de bienvenida, el cual también le indicara que puede empezar a escribir los mensajes que quiera enviar. El servidor manejara bitácora de las direcciones que se conectan como clientes, así como también de los mensajes que se recibieron.

C) CÁLCULOS Y REPORTE

Aplicación en ejecución.

5. RESULTADOS Y CONCLUSIONES

El alumno deberá presentar la práctica y definir las diferencias entre la comunicación con sockets y datagramas.

6. ANEXOS

7. REFERENCIAS

http://java.sun.com