Desarrollo de Sistemas Distribuidos - 4CM5

Tarea 5

Oscar Andres Rosas Hernandez
2014090642

UDP

El Protocolo de datagramas de usuario (UDP) utiliza el Protocolo de Internet (IP) para obtener una unidad de datos que también se denomina datagrama, de un dispositivo a otro a través de una red. UDP es un protocolo ligero definido en Request For Comments 768 en 1980. Se define como ligero ya que no requiere la pesada carga de tener detalles en un encabezado. Los anuncios de servicios, como las actualizaciones del protocolo de enrutamiento, la disponibilidad del servidor y las aplicaciones de streaming, como el vídeo y la voz, son algunos de los usos principales de UDP.

Para el UDP se utiliza un modelo de transmisión simple. Esto significa que no se puede garantizar la integridad o la fiabilidad de los datos, ya que proporcionan datagramas no seguros, fuera de servicio y, a veces, duplicados. El tráfico UDP, a diferencia del TCP, no requiere necesariamente una respuesta y no es necesario establecer una conexión para ser enviado.

Multicast

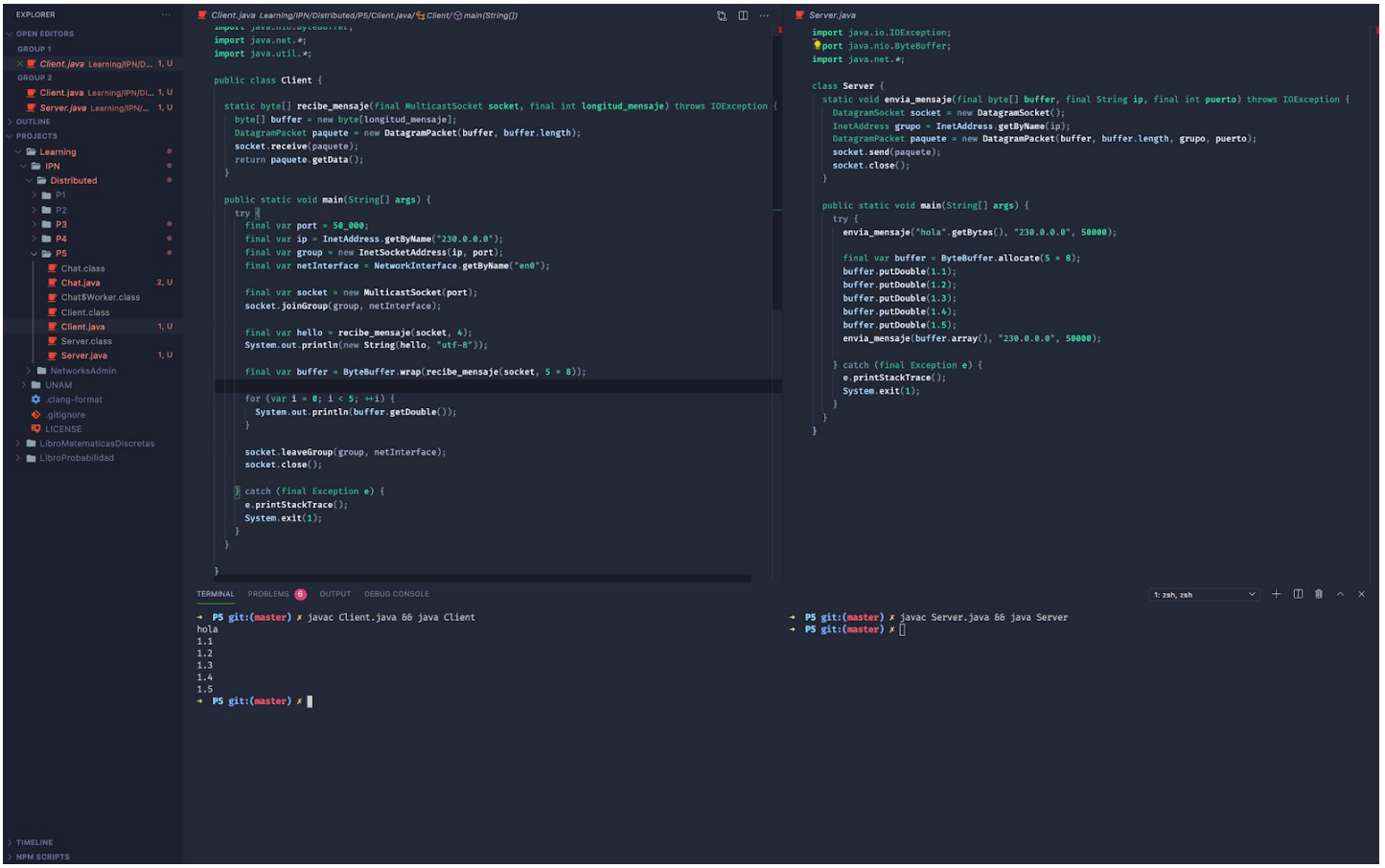
El intercambio de datos entre computadores es el objetivo principal de las comunicaciones en el campo de la informática. Inicialmente la comunicación implementada permitía sólo la transferencia entre un único origen y un único destino (comunicación uno-a-uno). La creciente evolución de las tecnologías de comunicación ha permitido alcanzar un progreso notable permitiendo la comunicación de un origen con múltiples destinos simultáneamente (comunicaciones uno-a-muchos y muchos-a-muchos). A esta forma de comunicación se le conoce como comunicación de grupo o alternativamente también conocida como multicast. La utilización y aprovechamiento de estas comunicaciones sigue siendo un tema de interés para los investigadores.

En la actualidad, la comunicación en una red de ordenadores es esencial para las aplicaciones, en especial para las aplicaciones distribuidas, tal como los sistemas de ficheros distribuidos. La comunicación multicast ha estado en el centro de interés en el área de Internet y ha contribuido en algunos éxitos importantes.

En infraestructura de red con IP (Internet Protocol), se usa IP multicast como método para comunicaciones uno-a-muchos y muchos-a-muchos. Los datos se envían sólo una vez, aunque lo reciban un número elevado de destinos. Los conmutadores de la red se encargan de replicar los datos (paquetes) por las salidas del conmutador necesarias para que alcancen todos los nodos del grupo de destinos. Se utiliza una dirección de grupo IP multicast. Lo utiliza tanto el origen como los destinos.

Desarrollo

La verdad la idea para implementar esta práctica es bastante sencilla, para empezar hice dos programas, uno como cliente y otro como servidor para probar que las dos funciones vistas en clase funcionaban.



Con estos funcionando pudimos escribir el programa principal de la práctica:

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

class Chat {
 static String name;

 static class Worker extends Thread {
   public void run() {
     // En un ciclo infinito se recibirán los mensajes enviados al grupo
     // 230.0.0.0 a través del puerto 50000 y se desplegarán en la pantalla.

     try {
       final var port = 50_000;
       final var ip = InetAddress.getByName("230.0.0.0");
       final var group = new InetSocketAddress(ip, port);
       final var netInterface = NetworkInterface.getByName("en0");

       final var socket = new MulticastSocket(port);
       socket.joinGroup(group, netInterface);

       while (true) {
         recibe_mensaje(socket, 2048);
         final var message = new String(data, "utf-8");
         final var otherName = message.split(":")[0];
         final var otherLine = message.split(":")[1];
         if (!otherName.equals(name)) {
           System.out.println(String.format("[%s] escribe: [%s]", otherName, otherLine));
         }
       }
     } catch (final Exception e) {
       e.printStackTrace();
       System.exit(1);
     }
   }
 }
```

```java
 static void envia_mensaje(final byte[] buffer, final String ip, final int puerto) throws IOException {
   DatagramSocket socket = new DatagramSocket();
   InetAddress grupo = InetAddress.getByName(ip);
   DatagramPacket paquete = new DatagramPacket(buffer, buffer.length, grupo, puerto);
   socket.send(paquete);
   socket.close();
 }

 static byte[] recibe_mensaje(final MulticastSocket socket, final int longitud_mensaje) throws IOException {
   byte[] buffer = new byte[longitud_mensaje];
   DatagramPacket paquete = new DatagramPacket(buffer, buffer.length);
   socket.receive(paquete);
   return paquete.getData();
 }

 public static void main(String[] args) throws Exception {
   Worker w = new Worker();
   w.start();
   String nombre = args[0];
   BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
   // En un ciclo infinito se leerá los mensajes del teclado y se enviarán
   // al grupo 230.0.0.0 a través del puerto 50000.

   Chat.name = nombre;
   while (true) {
     // System.out.println("Escribe el mensaje:  ");
     final var line = b.readLine();
     final var message = nombre + ":" + line;
     if (line != "") {
       envia_mensaje(message.getBytes(), "230.0.0.0", 50_000);
     }
   }
 }
}
```

Nota aquí importante tres cosas:

Primeramente que nada la interfaz de red esta "hardcodeada" para que funcionara en mi computadora, hay que cambiarla.

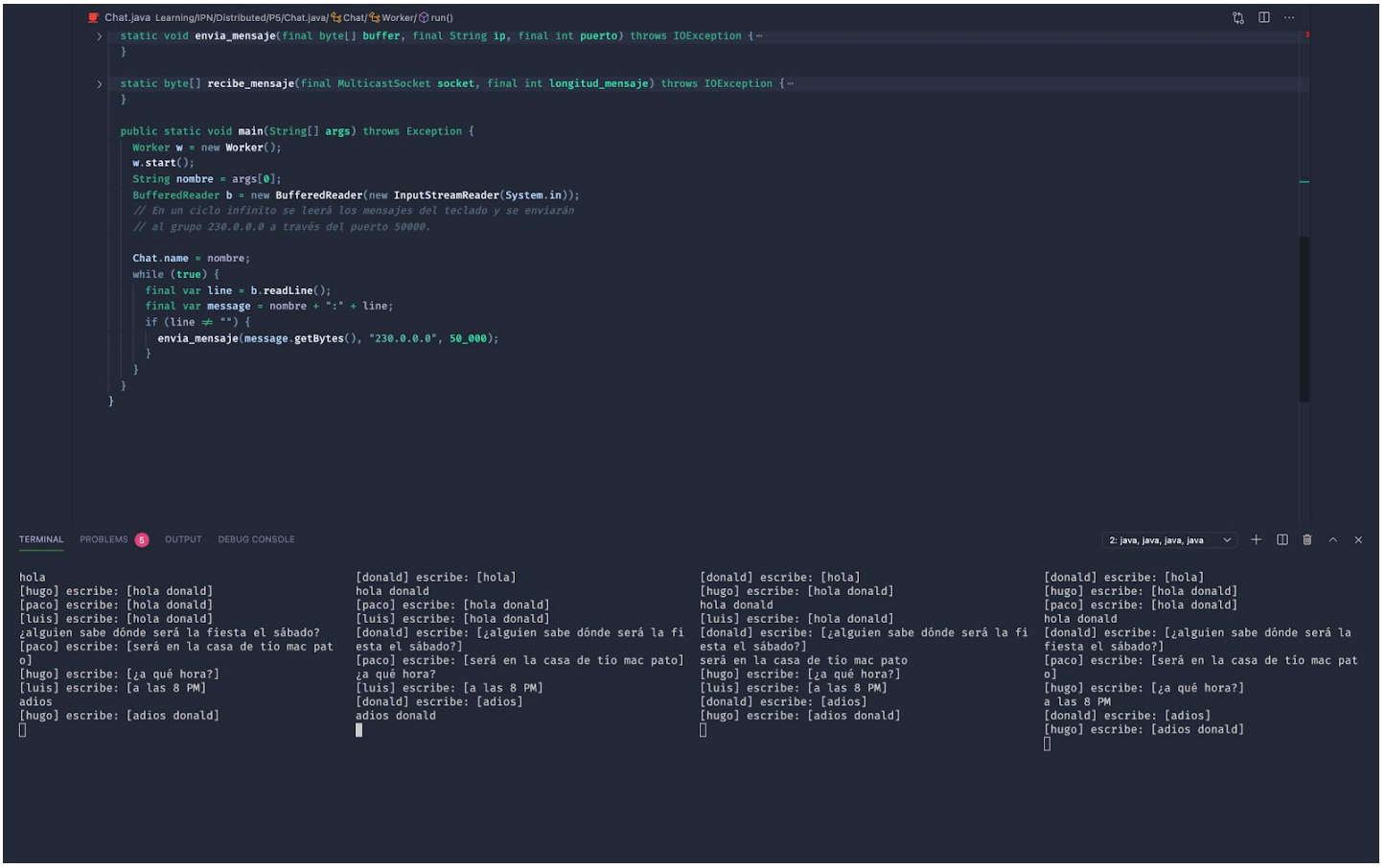
Igualmente desde Java 14 joinGroup esta deprecado a menos que también le pases explícitamente la interfaz de red.

Decidí dejar fuera por motivos estéticos el escribiendo mensajes, pero se puede volver a poner quitando el comentario.

Conclusiones

UDP es y los datagramas nos permiten de manera muy sencilla enviar mensajes a múltiples clientes de manera muy sencilla, pero es importante tener en cuenta (sobretodo en aplicaciones reales) que debido a la naturaleza de UDP ni el orden ni la unicidad de los mensajes están asegurados.

Evidencia


Chat.java Learning/IPN/Distributed/P5/Chat.java/Chat/Worker/run()
static void envia_mensaje(final byte[] buffer, final String ip, final int puerto) throws IOException {
}
static byte[] recibe_mensaje(final MulticastSocket socket, final int longitud_mensaje) throws IOException {
}
public static void main(String[] args) throws Exception {
Worker w = new Worker();
w.start();
String nombre = args[0];
BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
// En un ciclo infinito se leerá los mensajes del teclado y se enviarán
// al grupo 230.0.0.0 a través del puerto 50000.
Chat.name = nombre;
while (true) {
final var line = b.readLine();
final var message = nombre + ":" + line;
if (line != "") {
envia_mensaje(message.getBytes(), "230.0.0.0", 50_000);
}
}
}
}
TERMINAL
PROBLEMS 6
OUTPUT
DEBUG CONSOLE
2: java, java, java, java
hola
[hugo] escribe: [hola donald]
[paco] escribe: [hola donald]
[luis] escribe: [hola donald]
¿alguien sabe dónde será la fiesta el sábado?
[paco] escribe: [será en la casa de tío mac pat
o]
[hugo] escribe: [¿a qué hora?]
[luis] escribe: [a las 8 PM]
adios
[hugo] escribe: [adios donald]
[donald] escribe: [hola]
hola donald
[paco] escribe: [hola donald]
[luis] escribe: [hola donald]
[donald] escribe: [¿alguien sabe dónde será la fi
esta el sábado?]
[paco] escribe: [será en la casa de tío mac pato]
¿a qué hora?
[luis] escribe: [a las 8 PM]
[donald] escribe: [adios]
adios donald
[donald] escribe: [hola]
[hugo] escribe: [hola donald]
hola donald
[luis] escribe: [hola donald]
[donald] escribe: [¿alguien sabe dónde será la fi
esta el sábado?]
será en la casa de tío mac pato
[hugo] escribe: [¿a qué hora?]
[luis] escribe: [a las 8 PM]
[donald] escribe: [adios]
[hugo] escribe: [adios donald]
[donald] escribe: [hola]
[hugo] escribe: [hola donald]
[paco] escribe: [hola donald]
hola donald
[donald] escribe: [¿alguien sabe dónde será la
fiesta el sábado?]
[paco] escribe: [será en la casa de tío mac pat
o]
[hugo] escribe: [¿a qué hora?]
a las 8 PM
[donald] escribe: [adios]
[hugo] escribe: [adios donald]