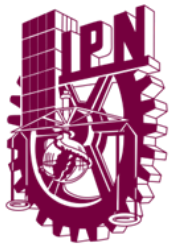


Instituto Politécnico Nacional

IPN

Escuela Superior de Cómputo

ESCOM

Inteligencia Artificial

6CV2

Práctica 3

“Sockets No Bloqueantes con UDP en Java”

Alumna:

Navarrete Becerril Sharon Anette

Fecha de entrega: “Lunes, 11-Noviembre-2024”

Profesor: Ojeda Santillan Rodrigo

OBJETIVO

El objetivo de esta práctica es implementar y demostrar el uso de sockets no bloqueantes orientados a datagramas utilizando el protocolo UDP (User Datagram Protocol) en un entorno cliente-servidor con el lenguaje de programación Java. Se busca que el estudiante sea capaz de enviar y recibir archivos de manera eficiente, utilizando operaciones de entrada y salida que no bloquean la ejecución del programa, optimizando así el manejo de múltiples conexiones en aplicaciones que requieren alta concurrencia, como la transmisión de archivos y servicios de mensajería.

INTRODUCCIÓN

UDP (User Datagram Protocol) es un protocolo de comunicación de nivel de transporte que se caracteriza por su simplicidad y eficiencia, ideal para aplicaciones que requieren alta velocidad y baja latencia. En el contexto de sockets no bloqueantes, UDP permite transmitir datos sin necesidad de establecer una conexión entre el cliente y el servidor, eliminando las verificaciones de entrega o confirmación de recepción de paquetes. Esta “comunicación sin conexión” significa que el emisor puede enviar paquetes de manera continua y rápida, sin esperar una respuesta del receptor, lo que reduce significativamente el tiempo de transmisión y mejora el rendimiento en aplicaciones donde la latencia es crítica.

Funciona enviando unidades de datos llamadas "datagramas". Cada datagrama incluye la información necesaria para que llegue al destino, pero no requiere un canal dedicado. Al no necesitar una conexión establecida, UDP permite que los datagramas se transmitan de forma independiente, optimizando así el rendimiento. Sin embargo, esta eficiencia tiene un costo: UDP no garantiza que los paquetes lleguen en el orden correcto ni que todos los paquetes enviados sean recibidos. En aplicaciones donde la velocidad es más importante que la fiabilidad (como en videojuegos en línea, servicios de transmisión en tiempo real o sistemas de mensajería instantánea), estas limitaciones de UDP son aceptables e incluso deseables, ya que cualquier retraso para asegurar la entrega podría afectar la experiencia del usuario.

Los sockets no bloqueantes permiten al programa continuar ejecutándose sin detenerse a esperar la finalización de las operaciones de entrada y salida. En el caso de UDP, esto es especialmente útil porque permite que un servidor reciba o procese datagramas de manera continua y rápida, gestionando múltiples clientes sin bloquearse en una sola operación. Esto es posible mediante la implementación de un “canal no bloqueante”, donde el programa puede iniciar una operación de envío o recepción y luego continuar con otras tareas en lugar de esperar a que los datos sean procesados. Con la ayuda de selectores (Selector en Java), el servidor puede verificar cuándo hay datagramas disponibles para ser procesados y atender a varios clientes en paralelo.
Al usar UDP en combinación con sockets no bloqueantes, las aplicaciones de baja latencia pueden lograr un rendimiento extremadamente eficiente. Esto permite que los datos se transmitan tan pronto como están listos, sin que la aplicación deba bloquearse en espera de cada operación de red.

DESARROLLO

Se implementaron dos programas en Java, uno para el cliente y otro para el servidor. Ambos utilizan DatagramChannel para el manejo de los sockets UDP no bloqueantes, y el flujo general de la comunicación sigue el siguiente esquema:

Servidor

El servidor utiliza un canal UDP (DatagramChannel) no bloqueante para recibir los datagramas enviados por el cliente. Este canal está configurado con un Selector que permite manejar múltiples operaciones de entrada/salida de forma eficiente.

Los datos recibidos se almacenan en un ByteBuffer y luego se escriben en un archivo local denominado "archivorecibido.txt" usando un WritableByteChannel.

El servidor permanece en un bucle infinito, esperando recibir datagramas de los clientes y procesándolos en tiempo real.

Código del servidor:

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.DatagramChannel;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.WritableByteChannel;
import java.net.InetSocketAddress;
import java.util.Iterator;

public class UDPFileServer {

    private static final int PORT = 8081;
    private static final int BUFFER_SIZE = 1024;

    public static void main(String[] args) {
        try {
            DatagramChannel serverChannel = DatagramChannel.open();
            serverChannel.configureBlocking(false);
            serverChannel.bind(new InetSocketAddress(PORT));

            Selector selector = Selector.open();
            serverChannel.register(selector, SelectionKey.OP_READ);

            ByteBuffer buffer = ByteBuffer.allocate(BUFFER_SIZE);
            WritableByteChannel fileChannel = new
FileOutputStream("archivorecibido.txt").getChannel();

            while (true) {
                selector.select();
                Iterator<SelectionKey> keys =
selector.selectedKeys().iterator();
  while (keys.hasNext()) {
                    SelectionKey key = keys.next();
                    keys.remove();

                    if (key.isReadable()) {
                        buffer.clear();
                        DatagramChannel channel = (DatagramChannel)
key.channel();
```

```
                        InetSocketAddress clientAddress =
(InetSocketAddress)
channel.receive(buffer);
                        buffer.flip();
                        fileChannel.write(buffer);
                        System.out.println("Archivo recibido desde: " +
clientAddress);
                    }
                }
            }

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

Cliente

El cliente abre un canal UDP (DatagramChannel) en modo no bloqueante y configura la dirección del servidor. Utiliza un ByteBuffer para almacenar los datos del archivo que se desea enviar.

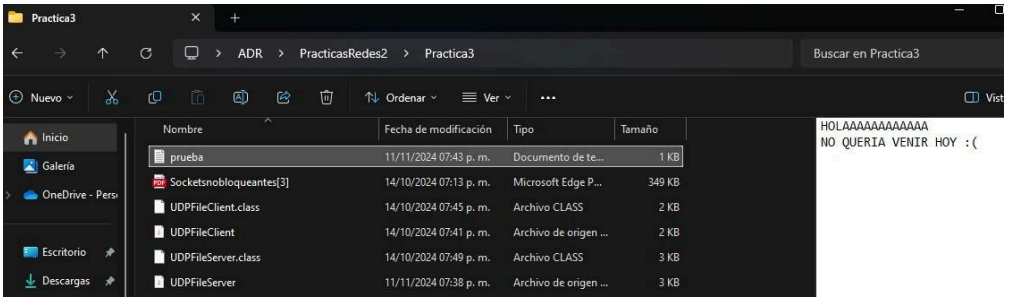
Los datos se leen del archivo en bloques y se envían al servidor mediante el método send(). Una vez que se completa el envío del archivo, el canal se cierra.

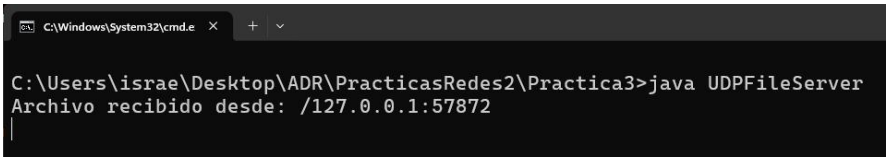
Código del cliente:

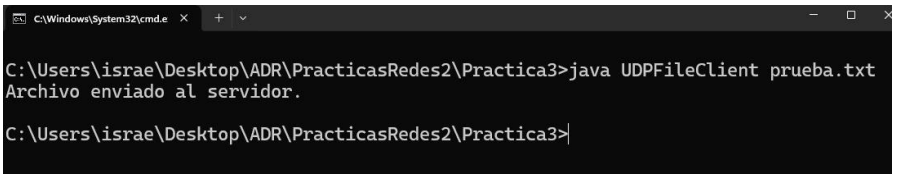
```
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.DatagramChannel;
import java.nio.channels.ReadableByteChannel;
import java.net.InetSocketAddress;

public class UDPFileClient {

    private static final int PORT = 8081;
    private static final int BUFFER_SIZE = 1024;

    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Usage: java UDPFileClient <file-path>");
            return;
        }

        String filePath = args[0];

        try {
            DatagramChannel clientChannel = DatagramChannel.open();
            clientChannel.configureBlocking(false);
            InetSocketAddress serverAddress = new
InetSocketAddress("localhost",
PORT);
```

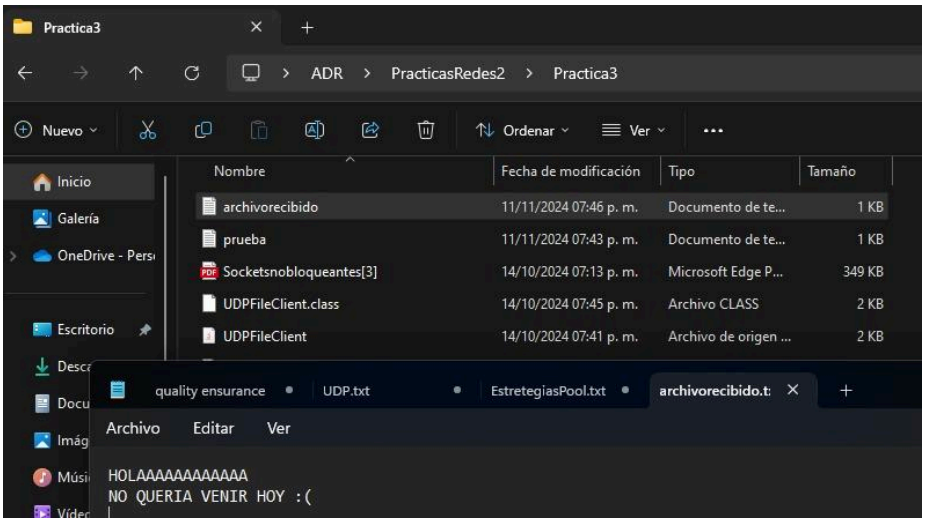
```
            ByteBuffer buffer = ByteBuffer.allocate(BUFFER_SIZE);
            ReadableByteChannel fileChannel = new
FileInputStream(filePath).getChannel();

            while (fileChannel.read(buffer) > 0) {
                buffer.flip();
                clientChannel.send(buffer, serverAddress);
                buffer.clear();
            }

            fileChannel.close();
            clientChannel.close();
            System.out.println("Archivo enviado al servidor.");

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

Capturas del funcionamiento:









CONCLUSIONES

Navarrete Becerril Sharon Anette:

La implementación de sockets no bloqueantes con el protocolo UDP permite manejar múltiples conexiones de manera eficiente sin que el programa se detenga a esperar la finalización de cada operación de entrada/salida. Este enfoque es particularmente adecuado para aplicaciones que requieren alta concurrencia y baja latencia, como la transmisión de archivos y sistemas de mensajería, dado que el canal no bloqueante permite al servidor procesar múltiples clientes en paralelo sin comprometer el rendimiento.