

# Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingenierías  
y Tecnologías Avanzadas



“UPIITA”

Ingeniería Telemática



Sistemas distribuidos

## Practica 1: Introducción al Manejo de Sockets

**Profesor:**

- Mata Rivera Miguel Félix

**Alumnos:**

- Guzmán Hernández Julio Cesar
- Rodríguez Venegas Thomas Francisco

**Grupo:** 2TM9

# Índice

<b>Objetivos .....</b>	<b>3</b>
<b>Requisitos .....</b>	<b>3</b>
<b>1. Comunicación con sockets localmente.....</b>	<b>3</b>
<b>2. Comunicación con sockets remotamente.....</b>	<b>5</b>
<b>3. Ejercicios de Tarea .....</b>	<b>7</b>
Servidor Java y Cliente en C .....	7
Servidor en Java.....	7
Cliente en C.....	10
Capturas del código en ejecución .....	11
Servidor En C y Cliente en Java.....	13
Servidor en C .....	13
Cliente en Java .....	14
Capturas de ejecucion del codigo .....	15
Retos para la resolución de estos problemas .....	16
<b>Link al repositorio de Github.....</b>	<b>16</b>

## **Objetivos**

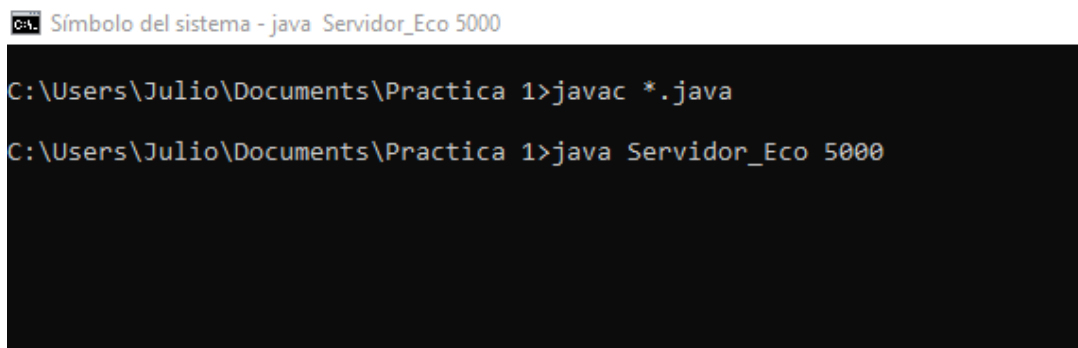
- Conocer cómo construir un Socket en Java, bajo la arquitectura CLIENTE-SERVIDOR y el enfoque de
- programación en red
- Comprender el concepto de arquitectura
- Comprender el concepto de transparencia en la comunicación
- Conocer el manejo básico de Github

## **Requisitos**

- Un ambiente de red (alámbrica o inalámbrica)
- Tener instalado y configurado java, un IDE (e.g. NetBeans o Eclipse)
- Privilegios de administrador para configuración de la red, y de las directivas de
- seguridad del sistema operativo [permisos de root en Linux]).

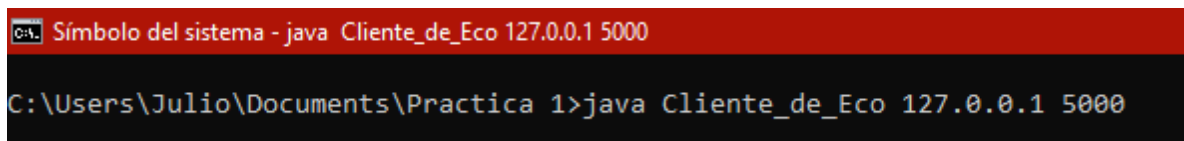
### **1. Comunicación con sockets localmente**

- a) Abra una terminal o ventana de comandos, compile todos los archivos java que descargo de Github y ejecute el programa Servidor\_Eco.java indicando el puerto de su preferencia, en este caso usamos el 5000



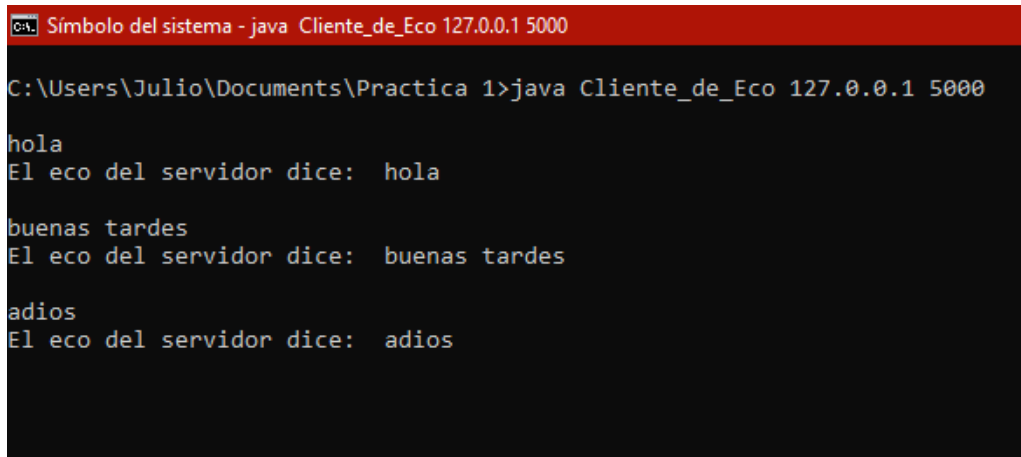
```
C:\Users\Julio\Documents\Practica 1>javac *.java
C:\Users\Julio\Documents\Practica 1>java Servidor_Eco 5000
```

- b) Abra otra terminal y ejecute el programa Cliente\_de\_Eco.java indicando: el puerto, la IP o el nombre del servidor (el nombre que tiene tu computadora, o puedes usar localhost que es universal para referirse a tu propia computadora)



```
C:\Users\Julio\Documents\Practica 1>java Cliente_de_Eco 127.0.0.1 5000
```

- c) Llama desde el cliente al programa Servidor Eco, esto se hace al escribir un mensaje de texto y dando enter. Deberás observar en la consola/terminal, la respuesta que te envió el servidor, es decir, al mensaje que envió [ que es precisamente el “eco” de lo que escribiste, (el mismo texto)].



```
Símbolo del sistema - java Cliente_de_Eco 127.0.0.1 5000

C:\Users\Julio\Documents\Practica 1>java Cliente_de_Eco 127.0.0.1 5000

hola
El eco del servidor dice: hola

buenas tardes
El eco del servidor dice: buenas tardes

adios
El eco del servidor dice: adios
```

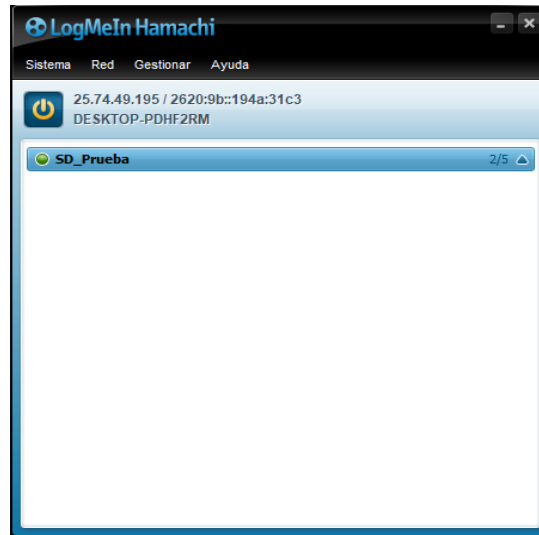
¿Qué necesitamos para que pueda comunicarse el programa Servidor (codificado en java) con un cliente (codificado en C) y viceversa?

R=

Para que se pueda realizar la conexión entre ambos lenguajes, se requiere que ambos programas puedan enviar datos en un formato estándar de red, puesto a que ambos manejan diferentes cantidades de bytes para sus variables, en el caso de C, existe una librería llamada htonl, que funciona como un middleware por así decirlo. Este permite poder transformar los datos que se envían y reciben a un formato de red estándar y se pueda enviar a cualquier otro programa en otro lenguaje.

## 2. Comunicación con sockets remotamente

Para este ejercicio se requiere que la ejecución se realice en una red WiFi o Ethernet, entre dos computadoras que pertenecen a la misma red o a diferentes redes, (si las redes son distintas, entonces debes poder indicar/gestionar la dirección IP de tu computadora como IP pública o en su defecto, puedes instalar un software de tipo Middleware para facilitar y lograr la comunicación entre tus programas (e.g. la herramienta Hamachi permite comunicar programas que están en diferentes redes)).



a) Ejecute el programa EchoServer en una computadora

```
Símbolo del sistema - java Servidor_Eco 5000

D:\escritorio\8vo semestre\Sistemas distribuidos\Practica1>java Servidor_Eco 5000
```

b) Ejecute el programa Cliente\_de\_Eco en la computadora B

```
Símbolo del sistema - java Cliente_de_Eco 25.74.37.94 5000

C:\Users\Julio\Documents\Practica 1>java Cliente_de_Eco 25.74.37.94 5000

hola
El eco del servidor dice: hola

conectado al servidor remoto
El eco del servidor dice: conectado al servidor remoto

adios
El eco del servidor dice: adios
```

¿Cómo se llama esta característica/funcionalidad en un sistema distribuido?

R=Compartir recursos

¿Qué es lo que permite que esta característica ocurra?

R= Para compartir los recursos de manera efectiva debe existir una entidad de software que se encargue de administrar el recurso y que presente una interfaz de comunicación por medio de la cual se pueda acceder a él de manera confiable y consistente.

### 3. Ejercicios de Tarea

Antes de realizar las conexiones entre servidores y clientes en los 2 lenguajes, Se define el servicio “cpp\_java” en el archivo encontrado en /etc/services, donde 15557 será el puerto de conexión.

#### Servidor Java y Cliente en C

Codifique dos programas usando sockets, en el enfoque cliente-servidor, que permita el intercambio de mensajes de texto .

El programa Servidor (debe ser codificado en java)

El programa Cliente (debe ser codificado en C)

Funcionamiento:

- Cuando se conecten entre sí, el cliente enviará una cadena de texto cualquiera, por ejemplo, Hola y el Servidor debe responder con algún otro mensaje, por ejemplo, Hola que tal.

#### Servidor en Java

Para empezar, lo primero que debemos de hacer es inicializar la conexión en el servidor java abriendo un socket con las siguientes líneas de código.

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

public class SocketServidor
{
    public static void main (String [] args)
    {
        new SocketServidor();
    }

    /**
     * Constructor por defecto. Hace todo lo que hace el ejemplo.
     */
    public SocketServidor()
    {
        try
        {
            ServerSocket socket = new ServerSocket (15557);
```

Posteriormente se espera la conexión de un cliente (en este caso codificado en C) que se conecte al mismo puerto del servidor usando la función `socket.accept()`. Si es exitosa se envía un mensaje de conexión aceptada y el cliente tendrá 10 segundos para mandar un dato.

```
System.out.println ("Esperando conexion del cliente \n");
Socket cliente = socket.accept();
System.out.println ("Conectado con cliente de " + cliente.getInetAddress());
System.out.println ("\nPuerto de Conexion: 15557\n");
cliente.setSoLinger (true, 10);
```

El servidor envía un mensaje al cliente que primero es procesado por una clase llama `DatoSocket`, para que pueda ser interpretado por el lenguaje C.

```
// Se prepara un dato para enviar.
DatoSocket dato = new DatoSocket("Hola Cliente");
```

```
import java.io.*;

public class DatoSocket implements Serializable
{
    /** Primer atributo, un int */
    public int c = 0;

    /** Segundo atributo, un String */
    public String d = "";

    public DatoSocket (String cadena)
    {
        if (cadena != null)
        {
            c = cadena.length();
            d = cadena;
        }
    }

    public String toString ()
    {
        String resultado;
        resultado = Integer.toString(c) + " -- " + d;
        return resultado;
    }

    public void writeObject(java.io.DataOutputStream out)
        throws IOException
    {
        // Se envía la longitud de la cadena + 1 por el \0 necesario en C
        out.writeInt (c+1);

        // Se envía la cadena como bytes.
        out.writeBytes (d);

        // Se envía el \0 del final
        out.writeByte ('\0');
    }
}
```



```

/**
 * Método que lee los atributos de esta clase de un DataInputStream tal cual nos los
 * envía un programa en C.
 * Este método no contempla el caso de que se envíe una cadena "", es decir, un
 * único \0.
 */
public void readObject(java.io.DataInputStream in)
throws IOException
{
    // Se lee la longitud de la cadena y se le resta 1 para eliminar el \0 que
    // nos envía C.
    c = in.readInt() - 1;

    // Array de bytes auxiliar para la lectura de la cadena.
    byte [] aux = null;

    aux = new byte[c]; // Se le da el tamaño
    in.read(aux, 0, c); // Se leen los bytes
    d = new String (aux); // Se convierten a String
    in.read(aux,0,1); // Se lee el \0
}
}

```

Con el dato procesado se prepara un flujo de salida de datos y se manda el mensaje al cliente.

```

// Se prepara un flujo de salida de datos, es decir, la clase encargada
// de escribir datos en el socket.
DataOutputStream bufferSalida =
    new DataOutputStream (cliente.getOutputStream());

// Se envía el dato.
dato.writeObject (bufferSalida);
System.out.println ("\nMensaje a enviar al cliente y su longitud: " + dato.toString());

```

Y después se prepara un flujo de entrada de datos para leer el mensaje del cliente y mostrarlo en la consola.

```

// Se prepara el flujo de entrada de datos, es decir, la clase encargada
// de leer datos del socket.
DataInputStream bufferEntrada =
    new DataInputStream (cliente.getInputStream());

// Se crea un dato a leer y se le dice que se rellene con el flujo de
// entrada de datos.
DatoSocket aux = new DatoSocket("");
aux.readObject (bufferEntrada);
System.out.println ("\nMensaje recibido por parte del cliente y su longitud: " + aux.toString());

cliente.close();

socket.close();
}
catch (Exception e)
{
    e.printStackTrace();
}
}

```

## Cliente en C

Para el cliente, inicializamos un socket para la conexión con el servidor, utilizando el siguiente código.

```
/*
 * Descriptor del socket y buffer para datos
 */
int Socket_Con_Servidor;
char cadena[100];
int longitud_Cadena;
int castRed;
/*
 * Se abre la conexión con el servidor, pasando el nombre del ordenador
 * y el servicio solicitado.
 * "localhost" corresponde al nombre de la computadora en la que
 * estamos ejecutando. Esta dado de alta en /etc/hosts
 * "cpp_java" es un servicio dado de alta en /etc/services
 */
Socket_Con_Servidor = Abre_Conexion_Inet ("localhost", "cpp_java");
if (Socket_Con_Servidor == 1)
{
    printf ("No puedo establecer conexión con el servidor\n");
    exit (-1);
}
```

Si la conexión no se puede iniciar, se manda un mensaje de error.

Posteriormente un mensaje “Hola servidor” y su longitud se envían al servidor para comprobar la conexión. Este mensaje debe ser transformado a un formato red con la función htonl para poder ser leído por java.

```
//Se va a enviar una cadena de 6 caracteres, incluido el \0
strcpy (cadena, "Hola servidor");
longitud_Cadena=strlen(cadena)+1;

//Antes de enviar el entero hay que transformarlo a formato red
castRed = htonl (longitud_Cadena);
Escribe_Socket (Socket_Con_Servidor, (char *)&castRed, sizeof(longitud_Cadena));
printf ("Longitud de la cadena enviada desde el cliente C:  %d\n\n", longitud_Cadena);

//Se envía la cadena
Escribe_Socket(Socket_Con_Servidor, cadena, longitud_Cadena);
printf ("El cliente C dice: %s\n\n", cadena);
```

Y por último se lee el mensaje enviado por el servidor, igual transformado a formato red para poder ser mostrado en consola.

```

//Se lee un entero con la longitud de la cadena, incluido el \0
Lee_Socket(Socket_Con_Servidor, (char *)&castRed, sizeof(int));
longitud_Cadena=ntohl(castRed);
printf("Se recibio un mensaje con longitud: %d\n\n", longitud_Cadena);

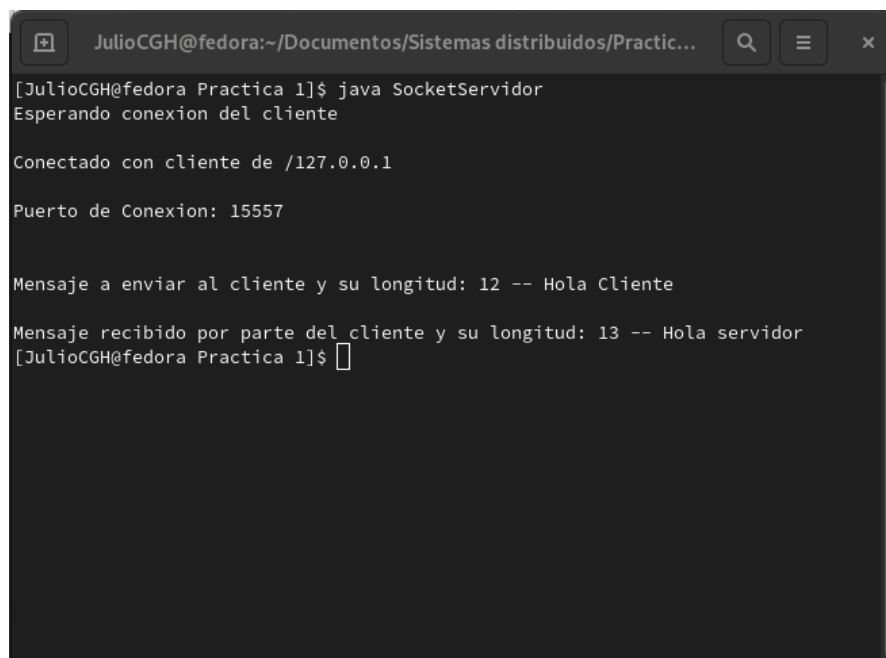
//Se lee la cadena de la longitud indicada
Lee_Socket(Socket_Con_Servidor, cadena, longitud_Cadena);
printf("El mensaje recibido por parte del servidor es: %s\n\n", cadena);

//Se cierra el socket con el servidor
close (Socket_Con_Servidor);
}

```

Tanto para la escritura, como la lectura se usaron las funciones Escribe\_Socket y Lee\_Socket.

### Capturas del código en ejecución



```

JulioCGH@fedora:~/Documentos/Sistemas distribuidos/Practic...
[JulioCGH@fedora Practica 1]$ java SocketServidor
Esperando conexion del cliente

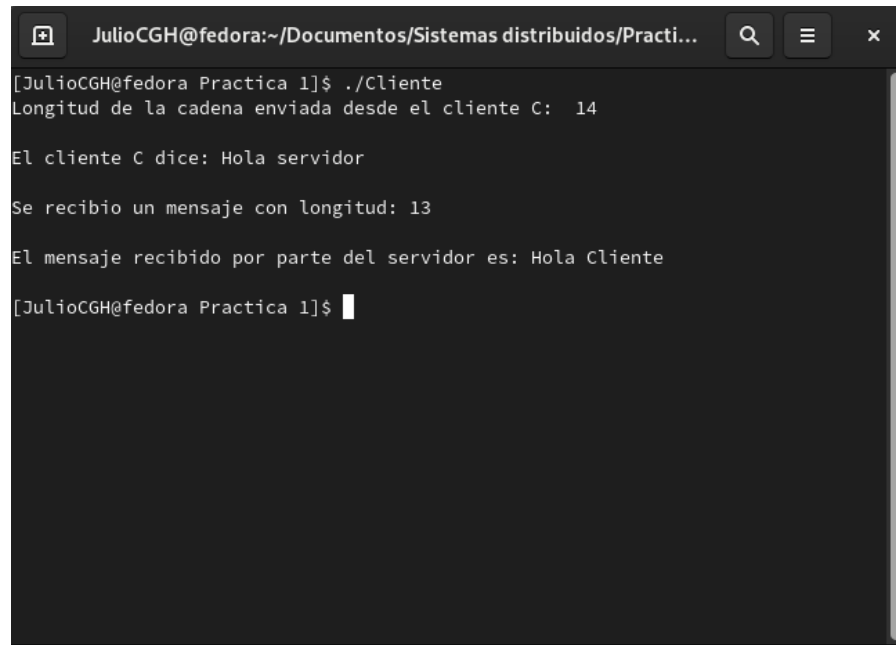
Conectado con cliente de /127.0.0.1

Puerto de Conexion: 15557

Mensaje a enviar al cliente y su longitud: 12 -- Hola Cliente

Mensaje recibido por parte del cliente y su longitud: 13 -- Hola servidor
[JulioCGH@fedora Practica 1]$

```



A terminal window titled "JulioCGH@fedora:~/Documentos/Sistemas distribuidos/Practi..." with search, menu, and close buttons. The terminal output shows a client program being executed, sending a message of length 14, receiving a response of length 13, and displaying the received message.

```
[JulioCGH@fedora Practica 1]$ ./Cliente
Longitud de la cadena enviada desde el cliente C: 14

El cliente C dice: Hola servidor

Se recibio un mensaje con longitud: 13

El mensaje recibido por parte del servidor es: Hola Cliente

[JulioCGH@fedora Practica 1]$
```

## Servidor En C y Cliente en Java

Codificar 2 programas usando sockets con el enfoque cliente-servidor, que permita que se envíen números enteros entre sí.

El programa Servidor (debe ser codificado en C)

El programa cliente (debe ser codificado en Java)

Funcionamiento:

- Cuando se conecten entre sí, el cliente enviará un entero y el servidor lo incrementará en uno. Ejemplo, el cliente envía un 5 y el servidor contestará con un 6, el programa terminará cuando el cliente escriba un cero.

### Servidor en C

Por parte del Servidor en C, se hace la conexión con la siguiente función:

```
Socket_Servidor = Abre_Socket_Inet ("cpp_java");
if (Socket_Servidor == -1)
{
    printf ("No se puede abrir socket servidor\n");
    exit (-1);
}
```

Posteriormente, se acepta la conexión con el cliente en Java.

```
Socket_Cliente = Acepta_Conexion_Cliente (Socket_Servidor);
if (Socket_Servidor == -1)
{
    printf ("No se puede abrir socket de cliente\n");
    exit (-1);
}
```

Si la conexión con el cliente es exitosa, se procede a realizar un ciclo While, en donde si el valor recibido por el cliente es 0, este termina su conexión. Dentro de este ciclo, se encuentran las funciones tanto de leer el socket del cliente, así como enviar los datos modificados hacia el cliente.

```
while(1){
    /*
     * Se lee la informacion del cliente
     */
    Lee_Socket (Socket_Cliente, (char *)&castRed,sizeof(int));
    Numero=ntohl(castRed);

    /*
     * Se escribe en pantalla la informacion que se ha recibido del
     * cliente
     */
    printf ("Soy Servidor, he recibido : %d\n", Numero);
    Numero=Numero+1;
    /*
     * Se usa htonl para enviar el numero modificado hacia el cliente Java
     */

    castRed=htonl(Numero);
    Escribe_Socket (Socket_Cliente, (char *)&castRed,sizeof(Numero));
    printf ("Soy Servidor, he enviado : %d\n\n", Numero);

    if(Numero==1){
        break;
    }
}
```

## Cliente en Java

El cliente recibe 2 parámetros, los cuales son la ip a conectar, así como el número de puerto

```
Run | Debug
public static void main(String[] args) throws IOException {

    if (args.length != 2) {
        System.err.println(
            "Uso desde consola: java Cliente_de_Eco <nombre de host (con
            System.exit(1);
    }

    String nombreHost = args[0];
    int numeroPuerto = Integer.parseInt(args[1]);
    new Cliente_de_Eco(numeroPuerto,nombreHost);
}
```

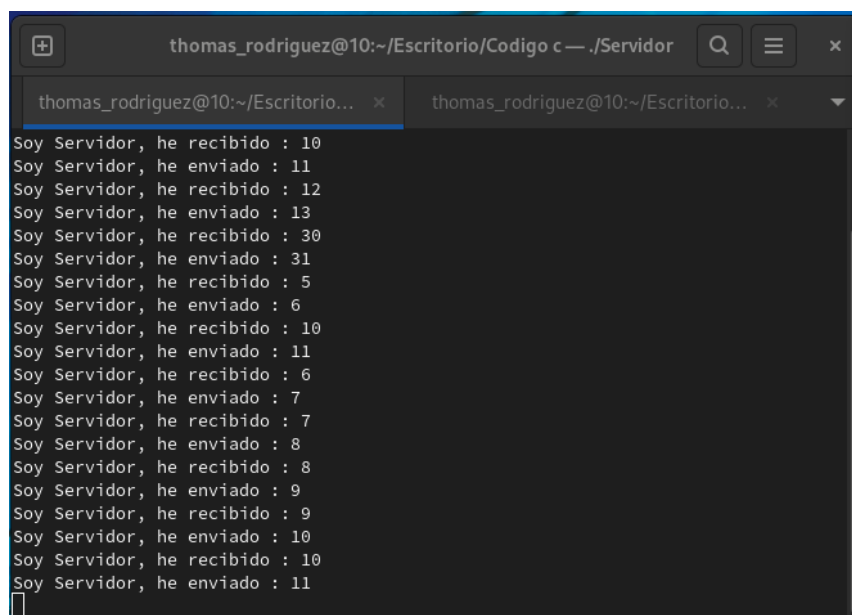
Se procede a realizar un ciclo while en donde se realiza el envío del entero

```
while (true){  
    System.out.println ("Ingrese el numero a enviar: ");  
    Numero=lectura.nextInt();  
  
    DatoSocket dato=new DatoSocket(Numero);  
  
    DataOutputStream bufferSalida=  
        new DataOutputStream(Server.getOutputStream());  
  
    //Se envia numero  
    dato.writeObject(bufferSalida);  
    System.out.println ("Numero enviado: "+dato.c);  
}
```

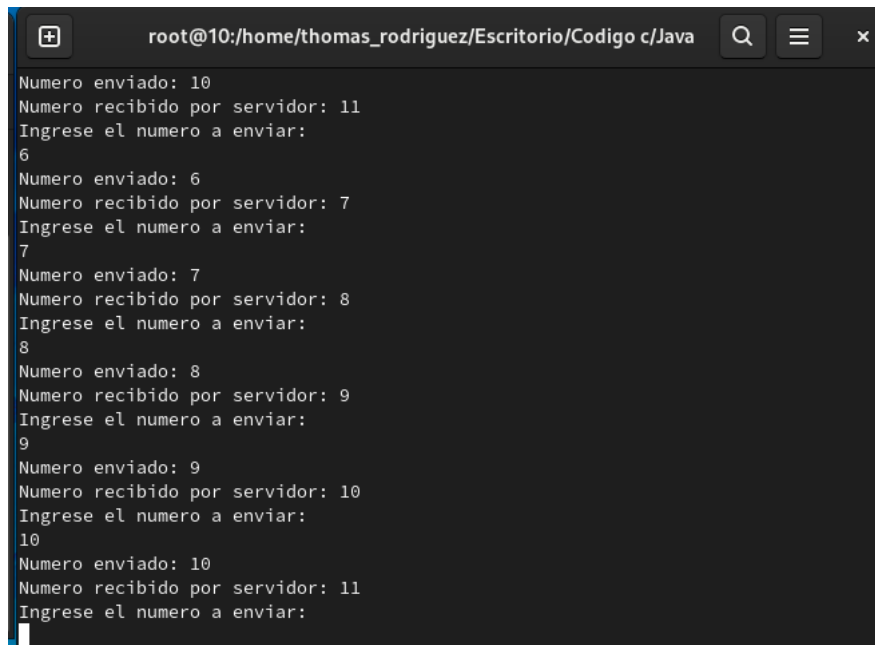
Posteriormente, se el cliente se queda esperando la respuesta el servidor, para así mostrar el incremento del entero que se envió.

```
//se recibe numero modificado  
DataInputStream bufferEntrada=  
    new DataInputStream(Server.getInputStream());  
  
DatoSocket aux =new DatoSocket[0];  
aux.readObject(bufferEntrada);  
System.out.println ("Numero recibido por servidor: "+aux.c);
```

### Capturas de ejecucion del codigo



```
thomas_rodriguez@10:~/Escritorio/Codigo c — ./Servidor  
thomas_rodriguez@10:~/Escritorio... x thomas_rodriguez@10:~/Escritorio... x  
Soy Servidor, he recibido : 10  
Soy Servidor, he enviado : 11  
Soy Servidor, he recibido : 12  
Soy Servidor, he enviado : 13  
Soy Servidor, he recibido : 30  
Soy Servidor, he enviado : 31  
Soy Servidor, he recibido : 5  
Soy Servidor, he enviado : 6  
Soy Servidor, he recibido : 10  
Soy Servidor, he enviado : 11  
Soy Servidor, he recibido : 6  
Soy Servidor, he enviado : 7  
Soy Servidor, he recibido : 7  
Soy Servidor, he enviado : 8  
Soy Servidor, he recibido : 8  
Soy Servidor, he enviado : 9  
Soy Servidor, he recibido : 9  
Soy Servidor, he enviado : 10  
Soy Servidor, he recibido : 10  
Soy Servidor, he enviado : 11  
█
```

A terminal window with a dark background and light text. The title bar shows the path 'root@10:/home/thomas\_rodriguez/Escritorio/Codigo c/Java'. The terminal displays a series of messages: 'Numero enviado: 10', 'Numero recibido por servidor: 11', 'Ingrese el numero a enviar:', followed by the input '6'. This sequence repeats with inputs 6, 7, 8, 9, and 10. The messages show the number received by the server, which is always one more than the number sent. The terminal has a search icon, a menu icon, and a close icon in the top right corner.

```
root@10:/home/thomas_rodriguez/Escritorio/Codigo c/Java
Numero enviado: 10
Numero recibido por servidor: 11
Ingrese el numero a enviar:
6
Numero enviado: 6
Numero recibido por servidor: 7
Ingrese el numero a enviar:
7
Numero enviado: 7
Numero recibido por servidor: 8
Ingrese el numero a enviar:
8
Numero enviado: 8
Numero recibido por servidor: 9
Ingrese el numero a enviar:
9
Numero enviado: 9
Numero recibido por servidor: 10
Ingrese el numero a enviar:
10
Numero enviado: 10
Numero recibido por servidor: 11
Ingrese el numero a enviar:
```

## Retos para la resolución de estos problemas

1. La diferencia entre plataformas
  - Cada lenguaje define de forma distinta sus variables, para poder solucionar esto, se usó la librería de htonl para convertir los datos de C a un formato estándar y así poder enviarse a otros lenguajes o plataformas
2. Uso de Máquinas virtuales
  - Tenemos experiencia en el uso de máquinas virtuales, pero, aun así, se requería de instalar paquetes adicionales, como es el caso del JDK de Java o algún IDE para el fácil manejo de los códigos
3. Bloque por parte del firewall
  - Cuando se realizó la conexión con Hamachi, un pc del equipo no permitía las conexiones, por lo cual se tuvo que editar configuraciones en el firewall de Windows para que nos permitiera hacer uso del cliente-servidor en Java

**[Link al repositorio de Github](#)**