

# Práctica 1

## Programación Distribuida y Tiempo Real

Lucas Di Cunzolo

**Abstract**—Introducción a Sockets en C y Java.

**Index Terms**—Programación Distribuida y Tiempo Real, c, java, L<sup>A</sup>T<sub>E</sub>X, sockets



### 1 FUNDAMENTOS TEÓRICOS

Primero se responderán los puntos relacionados con todos los conceptos teóricos necesarios para la comprensión de la práctica.

En C, se utilizará la librería estándar "**socket.h**", mientras que en Java, se va a utilizarán los paquetes "**java.net**", aunque utilizando ciertos paquetes para la interacción con el socket, provenientes de la librería "**java.io**", utilizando las clases **InputStream** y **OutputStream**.

#### 1.1 Similitudes y diferencias entre C y Java

Como primera y más grande diferencia entre ambos lenguajes, se puede remarcar la diferenciación entre "socket cliente" y "socket servidor", planteado por la librería de java. Lo que no sucede en C.

Para iniciar un socket en java, en modo servidor, se utiliza la clase **ServerSocket**, mientras que para iniciar uno en modo cliente, se utiliza la clase **Socket**.

En C, ambas creaciones quedan bajo la responsabilidad de la función **Socket()**, solo que para diferenciar el socket, el servidor debe "bindear" y quedarse a la espera de datos. Esto lo hace usando **Bind()** y **Listen()** respectivamente.

El cliente, se conectará a un socket utilizando la función **Connect()**

### 2 PUNTO 2

#### 2.1 A

2.1.1 *csock*

2.1.2 *javasock*

#### 2.2 B

2.2.1 *csock*

Se modificó el programa para aceptar varios tamaños de buffer (Ver apéndice A) Para probarlo rápidamente, se presenta la regla en make, lo que generará clientes y servidores que envían y reciben bytes

En este caso, la cantidad máxima de caracteres que se llegaron a leer fue de

```
# Genera todos los clientes y servidores
# 10^3 | 10^4 | 10^5 | 10^6 bytes
make all_buffers
```

```
# O generar uno específico
# Buffer de x bytes
make BUFFER_SIZE=x
```

```
# O generar solo buffer de:
# 10^3 bytes
make
```

```
# 10^4 bytes
make 10000
```

```
# 10^5 bytes
make 100000
```

```
# 10^6 bytes
make 1000000
```

### 2.2.2 *javasock*

## 2.3 C

### 2.3.1 *csock*

### 2.3.2 *javasock*

## 3 COMUNICACIÓN MEDIANTE SOCKETS TIPOS

En un programa C, para el envío de datos por un socket, se puede utilizar cualquier variable de tipo puntero. Con esto, se entiende que se puede utilizar hasta la misma variable que se utilizó para leer desde teclado (tipi `char*`).

Todo puntero en C, puede utilizarse de dos maneras.

- Manera estática:

```
int main()
{
    char buffer[256];
    // Uso de la variable buffer
    return 0;
}
```

Manejando memoria de manera estática, nos ahorramos el alocado de la memoria previo al uso, simplificando el código. Las principales desventajas son que si se necesita más de 256 caracteres, va a ser necesario cambiar de variable, y que de utilizar solamente 1 caracter, se tiene **siempre** alocado para 256 caracteres. La alternativa es utilizar memoria dinámica.

- Manera dinámica:

```
#include <stdlib.h>
int main()
{
    char *buffer = NULL;
    /* No se puede usar
       la variable buffer */
    buffer = (char*)malloc(256);
    /* Ya se puede usar
       la variable buffer */
    return 0;
}
```

En este caso va a ser necesario llamar a alguna de las funciones para alocado de memoria antes de utilizar la variable. De no hacer, generará errores.

La principal ventaja de utilizar memoria de esta manera, es la posibilidad de re-alocar su espacio en cualquier momento. Como principal desventaja es que se le complejidad al código.

Luego, el socket acepta como buffer una variable de tipo **void\***, lo que significa que se le puede enviar un puntero a cualquier tipo. El mayor problema que puede traer esto es que, como son bytes, el cliente no sepa como interpretarlos.

## 4 SERVIDOR DE ARCHIVOS

No solo sería posible, sino que ya existen 2 protocolos para envío de archivos por la red.

Estos protocolos se diferencian en, orientado a la conexión **FTP** o no orientado a la conexión **TFTP**.

### 4.1 FTP

Un servidor FTP se podría implementar utilizando 2 sockets para el servidor, uno utilizado para los datos, y otro utilizado para el control. (Puertos 20 y 21 en el estándar, respectivamente). Ambos socket deben ser del tipo TCP. Se cuenta con instrucciones específicas (muy similares a las de linux) para el uso de los archivos, como por ejemplo:

- **ls:** Lista los archivos.
- **cd:** Cambia de directorio.
- **close:** Cierra la conexión con el servidor remoto (TCP FIN).
- **delete:** Elimina un archivo.
- **get:** Baja un archivo.
- **mkdir:** Crea un directorio.

Entre otras instrucciones.

Todas las acciones se ejecutan **sobre** el filesystem del servidor remoto FTP.

### 4.2 TFTP

Un servidor TFTP utiliza solamente un socket, para realizar la conexión, del tipo UDP. Cuenta con un conjunto de instrucciones mucho más reducido que el servidor FTP.

### 4.3 Implementación propia

Para la implementación de un servidor, utilizaría

## 5 SERVIDORES CON Y SIN ESTADO

Una posible diferenciación de los servidores en el esquema cliente/servidor es la de servidor **con estado** y servidor **sin estado**

### 5.1 Servidor con estado

Un servidor con estado, es aquel que recuerda el estado en el que había quedado el sistema en la anterior conexión de un cliente, sin la necesidad de mantener la conexión. Esto quiere decir, por ejemplo, un servidor con estado es aquel que puede mantener una página web por ejemplo.

### 5.2 Servidor sin estado

Un servidor sin estado, es aquel que no almacena el estado entre conexiones de un cliente. Esto quiere decir, que el servidor **siempre** arranca en el mismo estado para todas las conexiones.

## APPENDIX A

### CÓDIGO C

Se presentarán secciones del código C de principal relevancia.

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#ifdef BUFFER_SIZE
#define BUFFER_SIZE 1000000
#endif

void error(char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[BUFFER_SIZE];
    if (argc < 3) {
        fprintf(stderr, "usage %s _hostname _port\n", argv[0]);
        exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr, "ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *) server->h_addr,
        (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);
```

```

serv_addr.sin_port = htons(portno);
if (connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0)
    error("ERROR_connecting");
//printf("Please enter the message: ");
bzero(buffer,BUFFER_SIZE);
//fgets(buffer,255,stdin);
int i;
for (i = 0; i < BUFFER_SIZE; i++) {
    buffer[i] = 'a';
}
n = write(sockfd,buffer,BUFFER_SIZE);
if (n < 0)
    error("ERROR_writing_to_socket");
bzero(buffer,BUFFER_SIZE);
n = read(sockfd,buffer,BUFFER_SIZE-1);
if (n < 0)
    error("ERROR_reading_from_socket");
printf("%s\n",buffer);
return 0;
}

```

## APPENDIX B

### CÓDIGO JAVA

Al código C

Se presentarán secciones del código Java de principal relevancia.

```

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

public class Client
{
    public static void main(String[] args) throws IOException
    {
        /* Check the number of command line parameters */
        if ((args.length != 2) || (Integer.valueOf(args[1]) <= 0) )
        {
            System.out.println("2 arguments needed: serverhostname port");
            System.exit(1);
        }

        /* The socket to connect to the echo server */
        Socket socketwithserver = null;

        try /* Connection with the server */
        {
            socketwithserver = new Socket(args[0], Integer.valueOf(args[1]));
        }
        catch (Exception e)

```

```

{
    System.out.println("ERROR_connecting");
    System.exit(1);
}

/* Streams from/to server */
DataInputStream fromserver;
DataOutputStream toserver;

/* Streams for I/O through the connected socket */
fromserver = new DataInputStream(socketwithserver.getInputStream());
toserver    = new DataOutputStream(socketwithserver.getOutputStream());

/* Buffer to use with communications (and its length) */
byte[] buffer;

/* Get some input from user */
Console console = System.console();
String inputline = console.readLine("Please_enter_the_message:_");

/* Get the bytes ... */
buffer = inputline.getBytes();

/* Send read data to server */
toserver.write(buffer, 0, buffer.length);

/* Recv data back from server (get space) */
buffer = new byte[256];
fromserver.read(buffer);

/* Show data received from server */
String resp = new String(buffer);
System.out.println(resp);

fromserver.close();
toserver.close();
socketwithserver.close();
}
}

```