



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

# Programación distribuida y de tiempo real

## Práctica 1 - Socket

*Alumnos:*

*Alan Fabian Castelli (726/9)*

*Franco Fico (1006/7)*

## Introducción

Se realiza la presentación de lo que es un modelo cliente/servidor realizado con sockets, en el cual se van a ir definiendo algunos conceptos, diferencias y similitudes, sobre la implementación tanto en java como en lenguaje C, obteniendo dicha información sobre la documentación de cada uno. A su vez, se realiza una seguidilla de ejercicios en el cual se ilustraron con imágenes y distintos tipos de pruebas para cubrir todos los conceptos y alcanzar el objetivo de la práctica de la mejor manera posible.

## Conceptos

A continuación se realizará un conjunto de definiciones en la cual son fundamentales tener en cuenta para realizar todo el resto de la práctica. Entre ellos podemos ver definiciones de documentación y ejemplos para su entendimiento.

### Modelo cliente/servidor

Desde el punto de vista funcional, se puede definir AL Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma. En el modelo cliente servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor (hace una petición), y este envía uno o varios mensajes con la respuesta (provee el servicio). En un sistema distribuido cada máquina puede cumplir el rol de servidor para algunas tareas y el rol de cliente para otras.

La idea es tratar a una computadora como un instrumento, que por sí sola pueda realizar muchas tareas, pero con la consideración de que realice aquellas que son mas adecuadas a sus características. Si esto se aplica tanto a clientes como servidores se

### Características

El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades de forma independientes.

Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.

Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.

La interrelación entre el hardware y el software están basados en una infraestructura poderosa, de tal forma que el acceso a los recursos de la red no muestra la complejidad de los diferentes tipos de formatos de datos y de los protocolos.

Su representación típica es un centro de trabajo (PC), en donde el usuario dispone de sus propias aplicaciones de oficina y sus propias bases de datos, sin dependencia directa del sistema central de información de la organización.

## Respuesta 1

### 1) Identifica similitudes y diferencias entre los sockets en C y en Java.

Tanto en C como en Java existe un método Socket.

En C, un socket se accede mediante un descriptor (número entero provisto por el sistema). El descriptor se retorna desde funciones o se envía a ellas.

Los clientes tienen los siguientes "métodos":

-Socket, devuelve un file descriptor que luego se puede usar para realizar llamadas al sistema. Si devuelve -1 hubo un error.

int socket (int dominio, int tipo, int protocolo)

– Dominio es AF\_UNIX o AF\_INET

- AF\_INET para procesos en distintos o el mismo host.
- AF\_UNIX únicamente para procesos en la misma máquina.

– Tipo es el estilo de la comunicación: SOCK\_STREAM o SOCK\_DGRAM.

– Protocolo permite especificar qué protocolo se usará:

- AF\_INET y stream requieren TCP
- AF\_INET y datagram requieren UDP
- Si se especifica 0, el sistema elegirá el más apropiado

Para realizar la conexión propiamente dicha, en C se utiliza el método connect:

```
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

- Especifica el socket local (sockfd) y el nombre del socket remoto
- El socket remoto debe haber sido relacionado a ese nombre
- Cuando la conexión tiene éxito, se establece y se podrá leer o escribir en el socket especificado por sockfd

En Java con el método Socket se realiza la conexión directamente, no hace falta utilizar otro método específico. éste método necesita como parámetros la IP y el puerto donde escucha el servidor.

En Java, en el momento que se requiere levantar (sea servidor o cliente) no se necesita de la existencia de uno o de otro, sino que son independientes entre sí hasta el momento de su conexión. A diferencia de C, este requiere que el servidor se encuentre conectado primero para que se establezca la comunicación, en el caso de no cumplir con dicha característica, se caerá el proceso cliente inmediatamente, comunicando un error.

Tanto en C como en Java existen métodos Read y Write, sin embargo, en Java no se lee o escribe directamente desde el socket sino que se necesitan InputStream y OutputStream.

En C, para implementar un servidor, para asociar un file descriptor a un puerto de la máquina, se utiliza el método bind. Con el método listen se esperan cierta cantidad de conexiones entrantes en un socket.

El método accept se usa para esperar las conexiones desde los clientes. Devuelve un nuevo socket que es donde se realiza la conexión. Es transparente al usuario. Es un nuevo File Descriptor

En Java, para implementar un servidor se utiliza el método ServerSocket(). Éste crea el socket donde se va a esperar la conexión con los clientes. Lleva el puerto donde se escucha como parámetro. Accept() mantiene el socket a la espera de una conexión

## Respuesta 2

**a.- ¿Por qué puede decirse que los ejemplos no son representativos del modelo c/s?**

Podría decirse que los ejemplos mostrados no son representativos de un modelo cliente/servidor dado que al realizar la lectura o escritura desde o hacia un socket, en realidad no se estaría realizando un envío o una recepción de petición, sino que solo se estaría realizando una comunicación

**b.- Muestre que no necesariamente siempre se leen/escriben todos los datos involucrados en las comunicaciones con una llamada read/write con sockets. Sugerencia: puede modificar los programas (C o Java o ambos) para que la cantidad de datos que se comunican sea de 10 3 , 10 4 , 10 5 y 10 6 bytes y contengan bytes asignados directamente en el programa (pueden no leer de teclado ni mostrar en pantalla cada uno de los datos del buffer), explicando el resultado en cada caso. Importante: notar el uso de “attempts” en “...attempts to read up to count bytes from file descriptor fd...” así como el valor de retorno de la función read (del man read).**

**c.- Agregue a la modificación anterior una verificación de llegada correcta de los datos que se envían (cantidad y contenido del buffer), de forma tal que se asegure que todos los datos enviados llegan correctamente, independientemente de la cantidad de datos involucrados.**

Para realizar la resolución del inciso, en primera instancia lo que se plantea es un modelo en el cual se transmiten de forma completa una variable de tipo String, la cual contendrá un conjunto de caracteres “A”.

Por un lado, para la resolución del lado Servidor, lo que se realizó es un bucle en el cual realiza un conteo de lo que ingresa por parte del cliente, al mismo tiempo de que realiza una suma parcial del tamaño de los datos que le van llegando (Esto ultimo da el resultado final del paquete recibido). Entonces el pseudocodigo de lo que se realiza es lo siguiente:

*Mientras (recibo los datos del canal != vacio)*

*devuelvo lo que recibí, realizado para la practica pero no necesario  
conteo de lo recibido + lo que tengo previamente.*

*Mostramos la nueva cantidad recibida.*

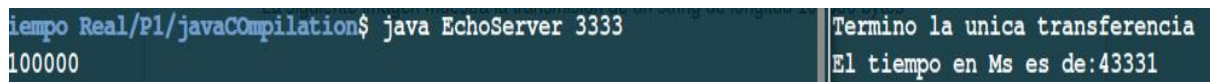
Por el lado del Cliente, vamos a considerar dos casos de análisis. Lo primero es mandar un string el cual va a contener distintos tipos de tamaños, autocompletados en el código para la práctica. Finalmente lo que hacemos es realizar una transmisión de forma completa y luego recibir un resultado del servidor para mostrar en pantalla dicha cadena. Bajo este planteo, el pseudocódigo es el siguiente:

```

Cargamos un string con A igual a la cantidad que se quiera transmitir
mientras no sea vacio lo que tenga que transmitir
    transmito lo que necesite
    imprimo lo que se recibe de servidor
pongo en null lo que tenga que transmitir
mostrar el tiempo que se tardo en transmitir // esto se realiza con una funcion de java

```

La siguiente imagen muestra la transmisión de un string de longitud 100000 bytes



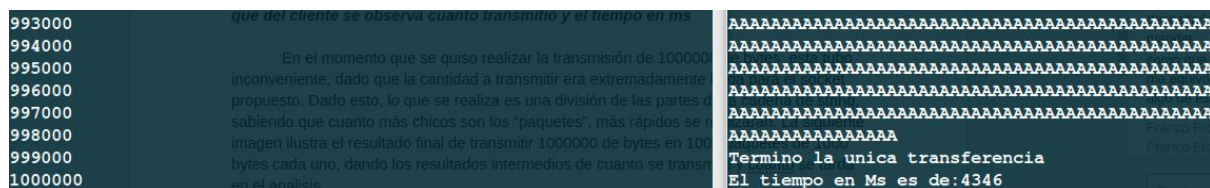
```

tiempo Real/P1/javaCompilation$ java EchoServer 3333
100000
Termino la unica transferencia
El tiempo en Ms es de:43331

```

**Figura 1: Del lado izquierdo el servidor con la cantidad de bytes recibidos, mientras que del cliente se observa cuanto transmitió y el tiempo en ms**

En el momento que se quiso realizar la transmisión de 1000000 de bytes, esta tuvo inconveniente, dado que la cantidad a transmitir era extremadamente larga para el socket propuesto. Dado esto, lo que se realiza es una división de las partes de la cadena de string, sabiendo que cuanto más chicos son los “paquetes”, más rápidos se realizarán. La siguiente imagen ilustra el resultado final de transmitir 1000000 de bytes en 1000 paquetes de 1000 bytes cada uno, dando los resultados intermedios de cuanto se transmite y cuánto se tarda en el análisis.



```

993000
994000
995000
996000
997000
998000
999000
1000000
Termino la unica transferencia
El tiempo en Ms es de:4346

```

**Figura 2: transmisión de 1000000 de bytes en paquetes de 1000 bytes.**

Para realizar el conteo del tiempo, lo que se implemento fue la clase `System.currentTimeMillis()`, pero desde el lado del cliente. Esto quiere decir que antes de empezar, primero se debe tener muy en cuenta arrancar/levantar el servidor, entonces el conteo arranca antes de que se realiza la conexion, entonces el tiempo final del mismo esta compuesto por :

$$\text{Tiempo final} = \text{Tiempo de conexion del cliente con servidor(desde que se levanta)} + \text{transmisión}$$

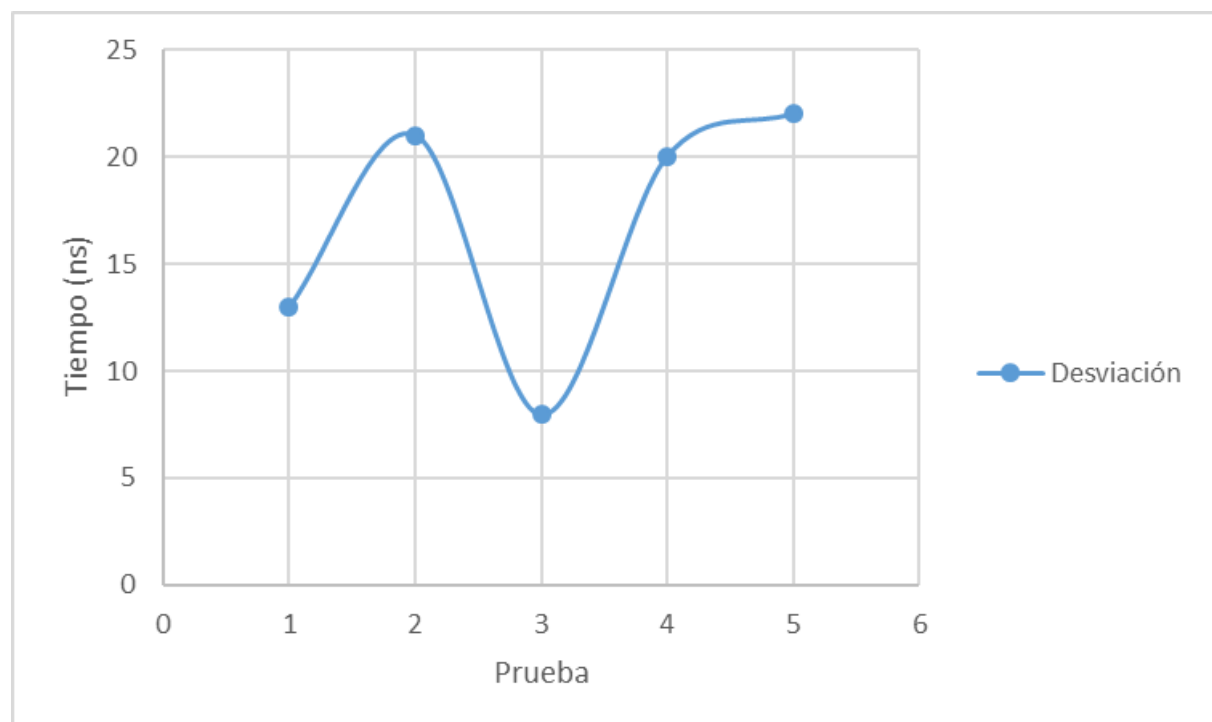
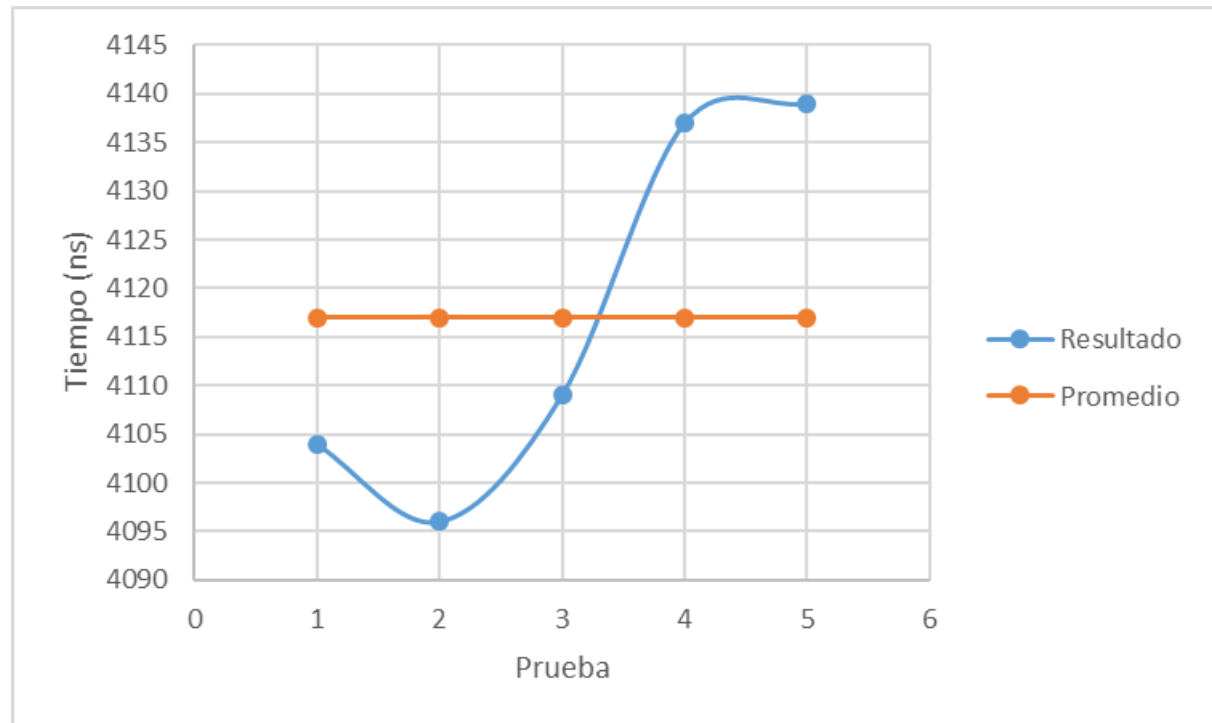
**d.- Grafique el promedio y la desviación estándar de los tiempos de comunicaciones de cada comunicación. Explique el experimento realizado para calcular el tiempo de comunicaciones.**

Se plantea una tabla en la cual se ejecuta varias veces el mismo código, en el cual vamos a poder contemplar que puede tardar más o menos según en el estado que se encuentra la conexión de socket entre ambos procesos. Para ser práctico, vamos a utilizar transmisión en partes ( de 10 paquetes) y transmitiendo de a 10000 bytes cada uno, dando un total de 100000 bytes.

Si realizamos el promedio de los 5 resultados, vamos a observar que dara un total (ms) = 4117. A partir de esto, la desviación estándar en cada punto es igual a la diferencia entre dicho promedio obtenido y el resultado en cada prueba, por lo que tendremos

Prueba	Desviación estándar
1	13
2	21
3	8

4	20
5	22





## Respuesta 3

**¿Por qué en C se puede usar la misma variable tanto para leer de teclado como para enviar por un socket? ¿Esto sería relevante para las aplicaciones c/s?**

En C, puede utilizarse la misma variable para leer de teclado como para enviar por un socket debido a que no es necesario usar un buffer intermedio para realizar la lectura o escritura, como es el caso de Java. Para aplicaciones cliente/servidor no es relevante, dado que no es necesario utilizar una misma variable, sino que lo que se requiere es el envío y recepción de peticiones. En el caso de utilizar la misma variable en C tiene el inconveniente de que se solapen los datos que se transmiten, dado que no se sabe en que momento se termino de mandar o recibir. Por el lado de java, como la implementacion esta dada por buffers intermedios que nos dan la seguridad de recibir y transmitir todo lo que un cliente o servidor desea, no se generan ningun tipo de solapamiento o perdida de informacion, sin importar la cantidad (hay que considerar que se puede romper la terminal si la misma es extremadamente larga, las pruebas dieron que en un millo, de bytes se rompe).

## Respuesta 4

**¿Podría implementar un servidor de archivos remotos utilizando sockets? Describa brevemente la interfaz y los detalles que considere más importantes del diseño. No es necesario implementar.**

Un servidor de archivos remotos es un tipo de servidor en el cual se permite la distribución y almacenamiento de archivos entre un cliente que se encuentre conectado en la misma red o canal de conexión. Para el desarrollo del mismo, el concepto de socket es suficiente si se quisiera plantear entre un par de computadoras (en la cual se tiene un servidor por un lado y un cliente por otro).

Como es de esperar, lo que se necesita es tener un solo servidor y varios clientes permitiendo que los mismos se puedan transmitir la información/archivos entre ellos. Bajo este concepto, se debe considerar:

- Plantear una comunicación del tipo socket entre cliente/servidor, respetando la lógica de dicho modelo.
- Como no se pueden conectar más de un cliente a la “Red”, se debe tener en cuenta el concepto Threads, en el cual por cada uno se debe plantear el servidor con un único cliente.
- Todos los pares cliente/servidor deben tener algún protocolo por el cual se comuniquen entre sí; de otro modo, los datos que pasan de un lado a otro no tendrían sentido. Este depende enteramente de la comunicación requerida por ellos para llevar a cabo la tarea.
  - *autenticación*: verificar la identidad del cliente.
  - *seguridad de datos*: para que estos no puedan ser accedidos inapropiadamente.
  - *privacidad*: garantizar que la información privada de un usuario, no sea accedida por alguien no autorizado.
  - *protección*: asegurar que las aplicaciones no monopolizan los recursos del sistema.
  - *autorización*: verificar si el cliente tiene acceso al servicio proporcionado por el servidor.
- Se debe tener en cuenta la cantidad de información a transmitir, recordar las pruebas realizadas en el ejercicio 2.

Teniendo en cuenta lo expresado anteriormente, el funcionamiento consistiría en lo siguiente:

1. Cierta cliente envía un mensaje al servidor indicando el archivo que necesita.
2. El servidor recibe el mensaje, lo procesa y busca el archivo solicitado
3. El servidor le envía un mensaje con el archivo al cliente que lo solicitó.

## Respuesta 5

**Defina qué es un servidor con estado (stateful server) y qué es un servidor sin estado (stateless server).**

Un servidor con estado es un servidor que recuerda el estado del cliente entre peticiones.

Un servidor sin estado no guarda ninguna información de los clientes.

Al usar un servidor sin estado, el cliente debe especificar los nombres completos de los archivos en cada petición, especificar la localización para leer o escribir, o re-autenticar para cada petición.

Al usar un servidor con estado, el cliente puede enviar menos datos con cada petición.