TRABAJO PRACTICO 1

Contenido

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

2) Tanto en C como en Java (directorios csock-javasock):

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

b- Muestre que no necesariamente siempre se leen/escriben todos los datos involucrados en las comunicaciones con una llamada read/write con sockets.

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.

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.

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?

4) ¿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?

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

Bibliografía

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

1) Sockets en C:

1- Se representan los sockets con file descriptor

Desde el lado del servidor se siguen los siguientes pasos:

- Apertura de un socket mediante la función socket()
- Se avisa al sistema operativo que abrimos un socket mediante la función bind()
- Se avisa al sistema para que atienda la conexión mediante la función listen()
- Se piden y aceptan las conexiones de clientes al SO mediante la función accept()
- Escribir y recibir datos del cliente mediante las funciones write() y read()

Desde el lado del cliente se siguen los siguientes pasos:

- Apertura de un socket mediante la función socket()
- Se solicita conexión con el servidor mediante la función connect()
- Escribir y recibir datos del servidor mediante funciones write() y read().
- Cerrar la comunicación mediante la función close().

Sockets en java:

Se representan los sockets con clases proporcionada por el paquete Java.net

Tanto desde el cliente como desde el servidor se siguen los siguientes pasos:

- Crear socket
- Abra el flujo de entrada / salida conectado a Socket
- Operación de lectura / escritura de Socket de acuerdo con cierto protocolo
- Cierre el socket.

A través de las clases del paquete java.net, los programas Java pueden utilizar TCP o UDP para comunicarse a través de Internet. Las clases URL, URLConnection, Socket, y SocketServer utilizan TCP para comunicarse a través de la Red. Las clases DatagramPacket y DatagramServer utilizan UDP.

La clase Socket del paquete java.net es una implementación independiente de la plataforma de un cliente para un enlace de comunicación de dos vías entre un cliente y un servidor. Utilizando la clase java.net.Socket en lugar de tratar con código nativo, los programas Java pueden comunicarse a través de la red de una forma independiente de la plataforma.

Además la clase ServerSocket implementa un socket que los servidores pueden usar para escuchar y aceptar conexiones a clientes. Las lecturas y escrituras al socket se resuelven utilizando input y output stream.

Ambos sockets utilizan protocolos TCP/UDP, y pueden escribir y recibir datos mediante funciones write y read.

2) Tanto en C como en Java (directorios csock-javasock):

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

2)

a- En el modelo C/S los servidores siempre se quedan a la espera de nuevas solicitudes, un ejemplo de esto es la red de Internet donde existen ordenadores de diferentes personas conectadas alrededor del mundo, las cuales se conectan a través de los servidores de su proveedor de Internet por ISP donde son redirigidos a los servidores de las páginas que desean visualizar y de esta manera la información de los servicios requeridos viajan a través de Internet dando respuesta a la solicitud demandada, en cambio en los ejemplos que tenemos el servidor recibe una sola solicitud y después cierra la conexión, por lo tanto no son representativos del modelo c/s.

Otro aspecto del modelo c/s es que los servidores contienen recursos que pueden ser accedidos a través de una interfaz que los usuarios pueden solicitar, lo cual no existe en los ejemplos dados.

Tampoco se tiene manejo de los errores que pueden ocurrir cuando enviamos un mensaje que no entre en el buffer.

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).

b- Se modificaron los codigos Client.java y Server.java, que ahora se encuentran en la carpeta 2b, la modificación que se hizo es que ahora el cliente envía 4 mensajes con un for con el tamaño indicado para cada uno. Después el servidor con su respectivo for recibe los mensajes con el read y ahí podemos observar si se leyeron los bytes esperados.

```
C:\Users\fermi\Desktop\Facultad\SegundoSemestre4to\ProgramacionDistribuida\Practica 1\javasock\2b>java Client.java local
host 3000
Bytes enviados: 1000
--------------------------------
Bytes enviados: 10000
--------------------------------
Bytes enviados: 100000
--------------------------------
Bytes enviados: 1000000
--------------------------------
Me llego tu mensaje
```

```
C:\Users\fermi\Desktop\Facultad\SegundoSemestre4to\ProgramacionDistribuida\Practica 1\javasock\2b>java Server.java 3000
Con 10^3:
Bytes leidos: 1000
Bytes esperados: 1000
--------------------------------
Con 10^4:
Bytes leidos: 10000
Bytes esperados: 10000
--------------------------------
Con 10^5:
Bytes leidos: 100000
Bytes esperados: 100000
--------------------------------
Con 10^6:
Bytes leidos: 131071
Bytes esperados: 1000000
--------------------------------
```

Como vemos el mensaje de 10^6 bytes el servidor solo lee 131071 de los 1000000 esperados.

Viendo que los bytes se envían de forma correcta el problema debe estar en la lectura de parte del servidor, por eso revise la documentación sobre syscall read, en la cual se explica que los bytes se comienzan a leer una vez se encuentre la información disponible y utiliza TCP para optimizar las conexiones porque la información que está mandando es mucha. Visto esto, lo que parece estar sucediendo es que el cliente envía N partes de la información en segmentos TCP, el read leerá la información que esté disponible pero puede que no toda lo esté porque el cliente no termino de enviarla entonces sucede algo como lo que paso en lo mostrado.

Un dato a tener en cuenta es que en la ejecución dentro del docker el problema surge con 10^5 bytes pero la razón es la misma que explique arriba, también supongo que debido a la arquitectura distinta al estarse ejecutando en el docker como si fuera “Linux” afecta en parte el resultado

```
root@d887129b42de:/pdytr/Practica 1/javasock/2b# java Server 3000
Con 10^3:
Bytes leidos: 1000
Bytes esperados: 1000
--------------------------------
Con 10^4:
Bytes leidos: 10000
Bytes esperados: 10000
--------------------------------
Con 10^5:
Bytes leidos: 65482
Bytes esperados: 100000
--------------------------------
Con 10^6:
Bytes leidos: 32741
Bytes esperados: 1000000
--------------------------------
```

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.

c- Se creó un loop donde se leerá el mensaje recibido y no terminará hasta que se haya leído completo, como también se utilizara un checksum para asegurar que los datos lleguen correctamente.

A la hora de enviar primero enviamos 4 bytes que contiene el tamaño del mensaje (protocolo Type Length Value, el cual trabaja con un message format que con 4 bytes indica la longitud del mensaje), luego los 16 bytes que contienen el hash creado con checksum MD5 (son 16 para este tipo de checksum) y por último el contenido del mensaje.

```
buffer = new byte[bufferSize];

// Llena el buffer
for (int j = 0; j < bufferSize; j++)
    buffer[j] = 1;

checksum = MD5Checksum.create(buffer);

// Escribe los 4 primeros bytes (tamanio del mensaje)
toserver.writeInt(bufferSize);

// Escribe 16 bytes (MD5 Checksum)
toserver.write(checksum, 0, checksum.length);

// Escribe el mensaje
toserver.write(buffer, 0, bufferSize);

System.out.println("Bytes escritos: " + bufferSize);
System.out.println("--------------------------------");
```

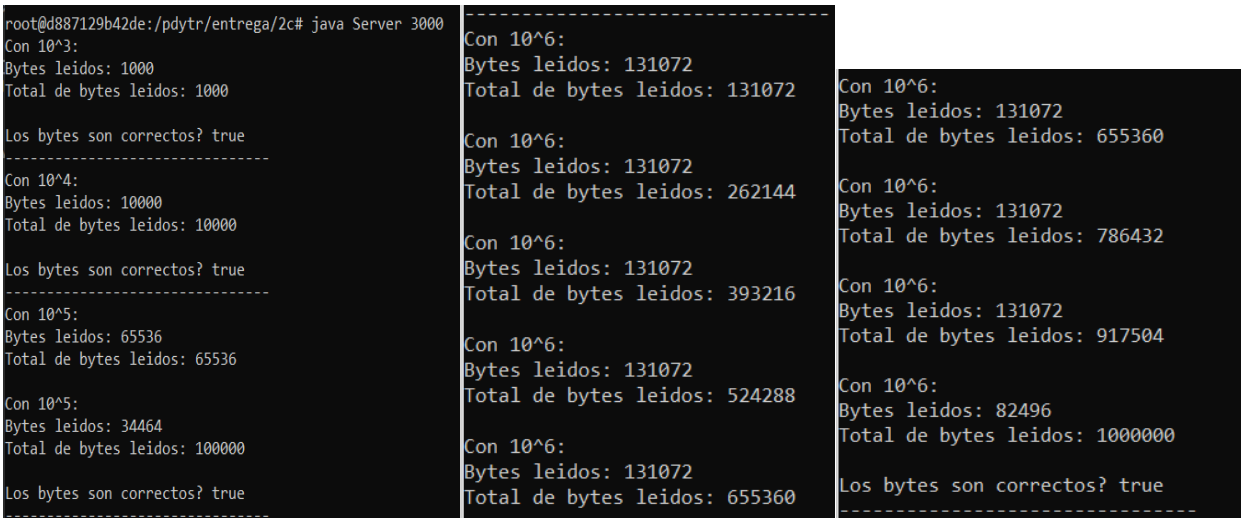
Primero se leen los 4 bytes que contienen tamaño del mensaje, los 16 que contienen el checksum y por último el mensaje.

Para poder leer todo el mensaje se creó una variable que almacena la cantidad de bytes leídos (readBytes) y otra para ir desplazándose sobre el mensaje (totalReadBytes), entonces a medida que leemos nos iremos desplazando sumándole al totalReadBytes los bytes leídos.

Una vez leído todo se fija los bytes son correctos comparando el checksum recibido con el mensaje leído.

```
int totalreadBytes = 0;

// Lee el mensaje hasta que complete el buffer
while (bufferSize > totalreadBytes)
{
    readBytes = fromclient.read(buffer,totalreadBytes,bufferSize - totalreadBytes);

    if ( 0 > readBytes)
    {
        System.err.println("Error al leer el buffer");
        System.exit(1);
    }

    totalreadBytes += readBytes;

    System.out.println("Bytes leidos: " + readBytes);
    System.out.println("Total de bytes leidos: " + totalreadBytes);
    System.out.println();
}

System.out.println("Los bytes son correctos? " + MD5Checksum.isValid(checksum, buffer));
System.out.println("--------------------------------");
```

Como se puede ver, en cada iteración donde el read no alcanzaba a leer todos los bytes de golpe, ahora se va recorriendo todo el mensaje segmento a segmento hasta leerlo por completo



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.

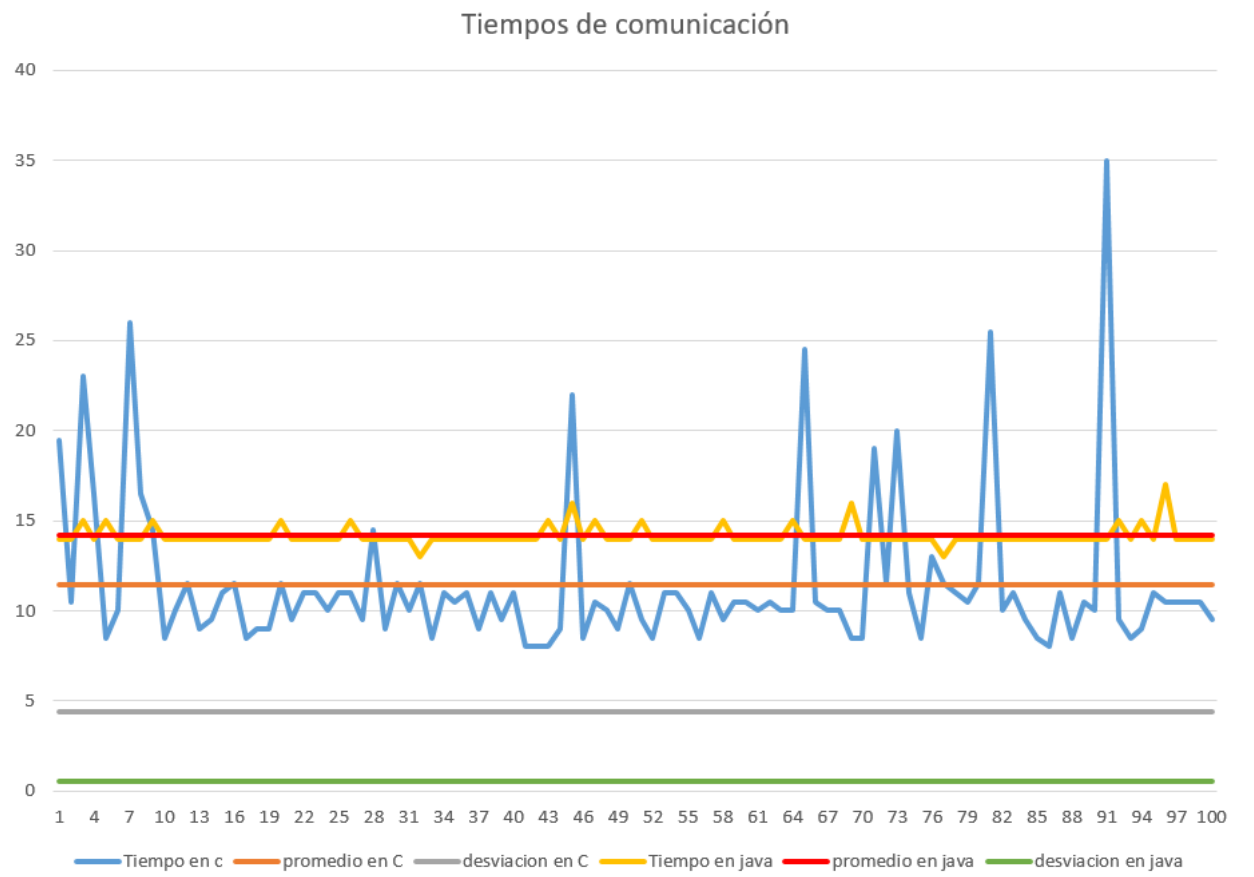
d- Ejecutando un código muy similar al hecho en el punto d con java y c (pero este sin checksum) se agregó lo siguiente:

Para poder calcular lo pedido hice que el cliente antes de enviar el mensaje empiece a contar el tiempo hasta recibir la respuesta del servidor, y contar el tiempo actual menos el que contamos cuando empezamos. Para calcular ese tiempo total debemos tener en cuenta las 2 comunicaciones que se hicieron (envió del cliente y respuesta del servidor), entonces si dividimos el tiempo calculado por esos 2 obtendríamos una estimación del tiempo total de comunicación. (Java a la izquierda y C a la derecha)

```java
// Empieza la cuenta para el tiempo de comunicacion
startTime = System.currentTimeMillis();

// Escribe los 4 primeros bytes (tamanio del mensaje)
toserver.writeInt(bufferSize);

// Escribe 16 bytes (MD5 Checksum)
toserver.write(checksum, 0, checksum.length);

// Escribe el mensaje
toserver.write(buffer, 0, bufferSize);

buffer = new byte[bufferSize];
// Espera la respuesta
fromserver.read(buffer, 0, bufferSize);

// Imprimo el tiempo de comunicacion total
System.out.println((System.currentTimeMillis() - startTime)/2);
```

```c
//Empieza la cuenta para el tiempo de comunicacion
timetick = dwalltime();

// Envia un mensaje al socket
n = write(sockfd,buffer,strlen(buffer));

if (n < 0)
     error("ERROR writing to socket");
bzero(buffer,bufferSize);

// Espera recibir una respuesta
n = read(sockfd, buffer, bufferSize);
if (n < 0) error("ERROR reading from socket");

close(sockfd);

// Se imprime el tiempo de comunicación
printf("%f\n", ( dwalltime() - timetick ) / 2);
```

Se creó un script (tanto para java como C) para poder obtener los números en un csv que use para crear el Excel y el siguiente gráfico con los tiempos de comunicación:



Como podemos apreciar parece que en el caso de java los tiempos varían mucho menos que en c, pero en promedio C es más rápido, algo que se nota a la hora de ejecutarlo en el docker.

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?

3) En la función write de C se necesita un puntero y la cantidad de datos a enviar aparte del file descriptor, la función fgets nos provee la posibilidad de leer un buffer de chars especificado y guardarlo como puntero en la variable que utilicemos, por ej la variable buffer del ejemplo de abajo, la cual usamos como puntero para el envío del socket en el write como se ve en este código:

```
int sockfd, portno, n;
struct sockaddr_in serv_addr;
struct hostent *server;
char buffer[256];
```

```
fgets(buffer,255,stdin);
n = write(sockfd,buffer,strlen(buffer));
```

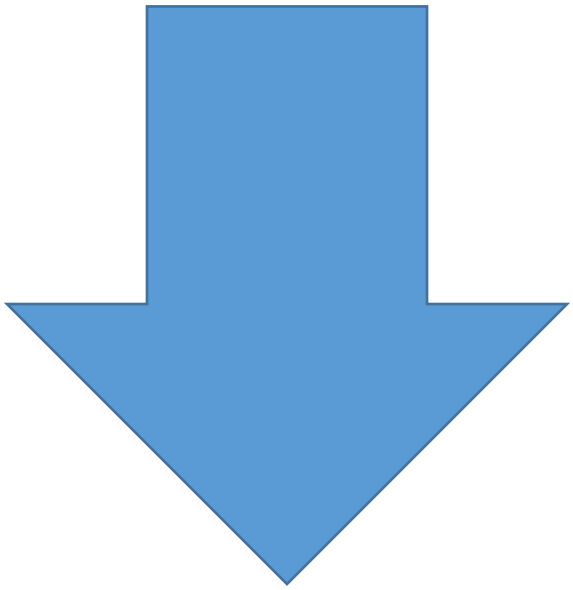
Esto nos permite alocar memoria una vez y usar el puntero para enviar por socket lo leído desde teclado, lo cual es relevante para las aplicaciones c/s porque podríamos hacer los pedidos de clientes en 2 líneas como se ve en el código mostrado

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.

4) Se puede hacer implementando el protocolo FTP.

En el caso de que se trate de subir (load) un archivo al servidor de archivos remotos, considero importante que el cliente y el servidor hagan lo siguiente:

Cliente	Servidor
1- Envía un pedido de subir un archivo como string	2- Recibe el archivo como string en un buffer con la misma cantidad de bytes enviados por el cliente
4- Recibe el mensaje con el booleano, en caso de recibir false cancela la subida y en caso de recibir true continua con los siguientes pasos	3- Busca si existe ese archivo, si no existe le envía un mensaje con true para que proceda con los siguientes pasos, si existe le manda un mensaje con false al cliente y se cancela la carga
5- Envía el tamaño del archivo	6- Recibe el tamaño del archivo en un buffer con los bytes necesarios
7- Envía un checksum del contenido y nombre del archivo en los bytes necesarios para ese checksum especifico	8- Recibe el checksum en un buffer de la misma cantidad de bytes enviados
9- Envía el contenido del archivo	10- Recibe el contenido del archivo
12- Recibe la respuesta del servidor	11- Calcula el checksum del nombre y contenido del archivo, luego compararlo con el recibido
	13- Si el checksum coincide se envía como respuesta un código de éxito y se guarda el archivo, sino se envía como respuesta un código de error y se descarta el archivo.



En el caso de que se trate de descargar (download) un archivo desde el servidor de archivos remotos, considero importante que el cliente y el servidor hagan lo siguiente:

Cliente	Servidor
1- Envía un pedido del archivo a descargar como string	2- Recibe el tamaño del archivo en un buffer con los bytes necesarios
4- Recibe el mensaje con el booleano, en caso de recibir false cancela la descarga y si recibe true continua con los siguientes pasos	3- Busca si existe ese archivo, si existe le envía un mensaje con true para que proceda con los siguientes pasos, si no existe le manda un mensaje con false al cliente y se cancela la descarga
5- Envia un mensaje avisando que continue con un booleano en true	6- Recibe el mensaje con booleano en true y procede con los siguientes pasos
8- Recibe el tamaño del archivo en un buffer con los bytes necesarios	7- Enviar el tamaño del archivo
10- Recibe el checksum en un buffer de la misma cantidad de bytes enviados	9- Enviar un checksum del contenido y nombre del archivo en los bytes necesarios para ese checksum especifico
12- Recibe el contenido del archivo	11- Enviar el contenido del archivo
13- Calcular el checksum del nombre y contenido del archivo, luego compararlo con el recibido	
14- Si el checksum coincide se guarda el archivo y sino se descarta	

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

5) Un servidor con estado es cuando el servidor recuerda el estado del cliente en su anterior conexión sin estar manteniendo una conexión, esto sería por ej cualquier servidor web donde logueaste y tu sesión quedo guardada, entonces cuando volves a entrar seguís logueado.

Un servidor sin estado es cuando no se mantiene ningún estado del cliente anterior, cada pedido es independiente del anterior.

Bibliografía

http://www.chuidiang.org/clinux/sockets/sockets_simp.php#introduccion

https://www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf

https://programmerclick.com/article/3591173590/

https://es.wikipedia.org/wiki/Cliente-servidor#Caracter%C3%ADsticas

https://linuxhint.com/read_syscall_linux/

https://www.baeldung.com/java-inputstream-server-socket

https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.md5?view=net-5.0

https://www.tutorialspoint.com/c_standard_library/c_function_fgets.htm

https://source.raysync.io/news/what-is-ftp-server-and-how-does-it-work

https://stackoverflow.com/questions/304268/getting-a-files-md5-checksum-in-java

https://stackoverflow.com/questions/27694797/md5-checksum-from-input-stream

https://stackoverflow.com/questions/27641464/generate-md5-string-in-java/27641555

https://stackoverflow.com/questions/36924598/understanding-htonl-and-ntohl

https://askubuntu.com/questions/1202067/bash-script-r-command-not-found

https://serverfault.com/questions/196734/bash-difference-between-and-operator

https://stackoverflow.com/questions/9834086/what-is-a-simple-explanation-for-how-pipes-work-in-bash

https://stackoverflow.com/questions/23489934/echo-2-some-text-what-does-it-mean-in-shell-scripting

https://www.ftptoday.com/blog/how-does-an-ftp-server-work-the-benefits