Programación Distribuida y Tiempo Real

Práctica 1

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

Un socket en C es accedido a través de un descriptor, el cual es obtenido o pasado a funciones de forma similar a los descriptores utilizados en archivos (file descriptors).
Para utilizar sockets es necesario importar <sys/sockets.h>.
Se puede crear un socket con la función:

int socket(int domain, int type, int protocol)

Con los argumentos se pueden configurar el dominio (local o a través de la red), el tipo de la comunicación (por paquetes o mediante conmutación de circuitos) y el protocolo (TCP/UDP).
Para poder utilizar los sockets se les otorga un nombre mediante la función.

int bind(int sockfd, struct sockaddr *my_addr, socklen_t, addrlen)

que le asignará un nombre al socket para poder ser referenciado desde otra computadora. Este nombre será un path en el filesystem en caso de ser tipo local o una dirección IP en caso contrario. Este nombre será referenciado por

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

para realizar una coneccion entre 2 computadoras enviando un socket (sockfd) al nombre de socket indicado en sockaddr. Cuando finaliza la coneccion se puede empezar a enviar y recibir datos.
Para comenzar a escuchar por nuevas conexiones:

int listen(int s, int backlog)

Las conexiones entrantes se almacenarán en una cola con tamaño backlog.
Luego se pueden aceptar las conexiones con:

int accept (int s, struct sockaddr *addr, socklen_t *addrlen);

que devolverá un socket para comunicarse con el cliente.

Para realizar la comunicación se utilizan:

ssize_t read (int fd, void *buf, size_t count);

ssize_t write (int fd, const void *buf, size_t count);

int send (int s, const void *msg, size_tlen, intflags);

int recv (int s, void *buf, size_t len, int flags);

A diferencia de C, Java oculta muchos detalles de cómo se realiza la conección.
Posee 2 clases: Socket y ServerSocket, dependiendo si van a ser utilizadas para el lado del cliente o del servidor respectivamente.
ServerSocket creará una instancia de socket cuando acepte una conexión mediante el mensaje accept().
Para usar los sockets basta con instanciarlos
new Socket(<hostname>, <port>);
new ServerSocket(<port>);
Una vez instanciadas se realiza la conexión según los parámetros pasados al constructor.
Luego podra realizarse la comunicación mediante getOutputStream() y getInputStream().

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?

Pära que sea Cliente/Servidor se deben cumplir 3 pasos:
Inicialización
Envió/recepción de peticiones
Finalización
En csock, no se cumple la finalización de la conexión. Para finalizar la conexión se puede utilizar close() o shutdown().
Además, tanto en csock como en javasock, solo se realiza una recepcion y envio de datos por parte del servidor, el cual no atiende a más de 1 cliente.

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).

En C:
10^3: se reciben todos los datos
10^4: se reciben todos los datos
10^5: se reciben 21845 o 10^5 datos
10^6: se reciben entre 21845 y 10^6 datos
read intentara leer hasta SSIZE_MAX, en caso de que la cantidad a querer leer sea mayor dependerá de la implementación. En Linux transferirá 2147479552 bytes a lo sumo.

En Java:
10^3: se reciben todos los datos
10^4: se reciben todos los datos
10^5: se reciben todos los datos
10^6: se reciben 131072 datos
En Java se transmiten todos los datos excepto en 10^6, segun la documentacion read intentara leer hasta el tamaño del buffer, pero podría leer menos.

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.

En Java:
Se crearon 2 buffers nuevos. Uno para recibir y enviar números, y otro para recibir y enviar partes del string.
Luego de iniciar la conexión, el cliente le envía al servidor el tamaño que tendrá el string que va a mandar.
El servidor devolverá el número 0 indicando que recibió el mensaje y que en este momento tiene 0 bytes recibidos del string.
El cliente intentará mandar una cantidad predefinida de bytes al servidor con 4 bytes que forman un entero indicando cuando le falta para terminar de mandar, este entero será usado por el servidor para verificar la correctitud de los datos recibidos.
El servidor devolverá cuantos datos logró acumular, de esta manera, cada vez que el servidor le manda una cantidad de datos menor a la acordada en el primer mensaje, el cliente envía datos recortando la parte ya enviada de forma de no enviarla 2 veces.

Una vez que el servidor envía al cliente una cantidad de datos acumulados igual a la cantidad acordada, el cliente saldrá del loop y dejará de enviarle datos.

En C:
Se poseerán 2 nuevos buffers en el servidor: uno para almacenar los datos parciales recibidos y otro para almacenar la cantidad totales de datos a pasar.
Se iniciará la conexión y el cliente le enviará al servidor la cantidad de datos que poseerá el string.
Una vez hecho esto comenzará a enviar los datos al servidor de a una cantidad predefinida de bytes según hagan falta (se realiza un corrimiento según se van enviando datos) llevando una variable que contará la cantidad de datos enviados utilizando un read que recibirá la cantidad de datos recibidos por el servidor. Una vez que esta variable llegue a ser la cantidad de datos totales dejará de enviar. En cada mensaje llevará la cantidad de datos mandados como medida de seguridad, el servidor, tendrá que separar los primeros 4 bytes para formar un entero que le indicará si ha recibido bien los datos.
El servidor, por su parte, recibirá el tamaño total del string, que utilizara para crear los 2 buffers, uno para el string completo y otro para las partes que vaya recibiendo. Mientras no reciba todos los caracteres del string, leerá del socket lo que le falte en el string parcial y lo concatena al string completo. Una vez terminan de pasar todos los datos el string completo queda en el arreglo buffer.

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 utilizó pandas para realizar los graficos. Para obtener los datos se calcularon los tiempos en las llamadas a write y read y se imprimieron en un archivo csv para ser procesados.
Se tomo el tiempo de la siguiente forma:
-En C:
Se utiliza la librería sys/time.h para obtener el tiempo en milisegundos.
-En Java:
Se utiliza System.nanoTime(); para obtener el tiempo en nanosegundos.

En ambos se realizo la medicion de la siguiente forma:
tiempo = tomarTiempo();
escribir;
leer;
imprimir(“label,” +((tomarTiempo()-tiempo)/2))

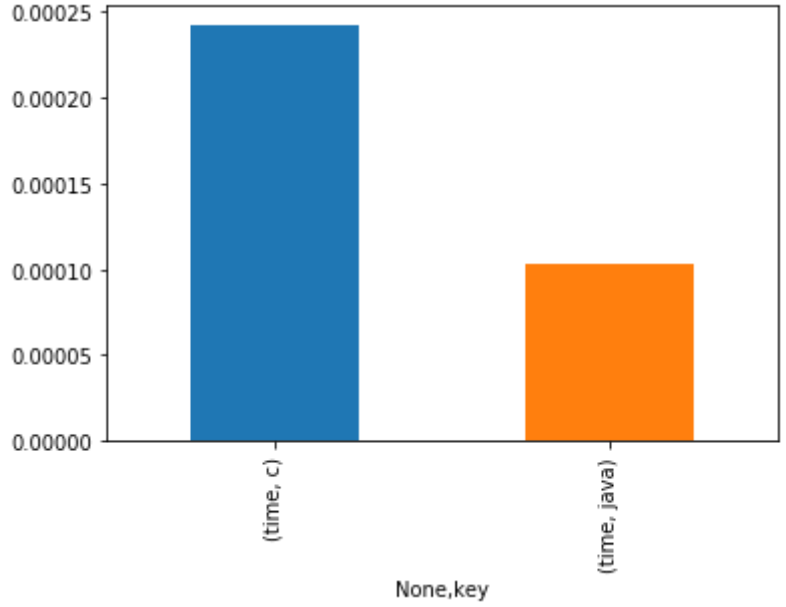
Se divide por 2 para indicar el tiempo que tarda en llegar el mensaje y en volver.

std c

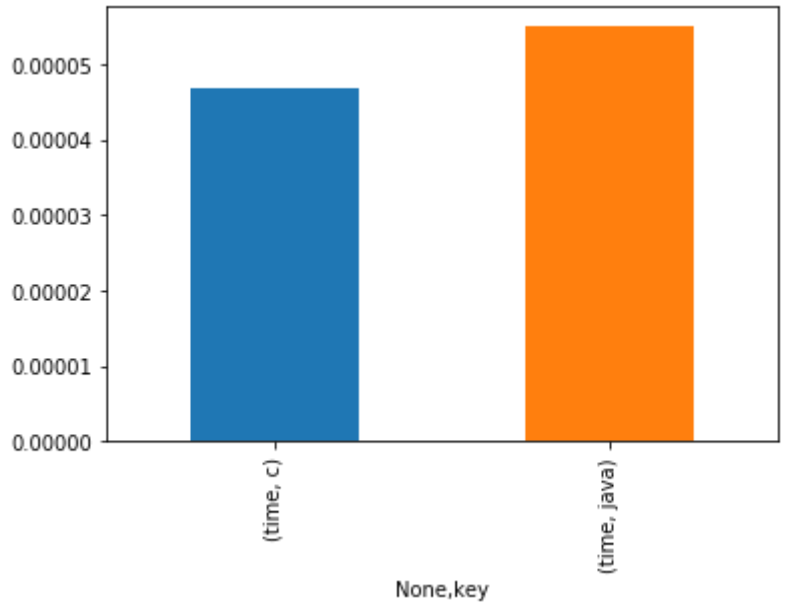
4.6826177604956284e-05
mean
0.00024190909090909091

std java
5.4987727513522956e-05
mean
0.00010288183333333333

mean



STD



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?

Esto es debido a que en C, cuando se lee el teclado se obtiene un array de caracteres de 1 byte cada uno, el formato necesario para poder enviar datos por socket.
En una aplicacion cliente servidor puede ser util para que el cliente le envie datos al servidor de forma mas sencilla de programar.

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.

Si. Para realizarlo, el cliente hará requests al servidor indicando con un mensaje el archivo que quiere obtener. Una vez el servidor reciba este mensaje, cargará el archivo y empezará a mandar bytes al cliente de forma similar a la especificada en el ejercicio 2.c para evitar que se pierdan datos (pero al revez, ya que ahora el servidor es el que manda los datos). Así mismo, el cliente puede requerir subir archivos, por lo que enviará al servidor el nombre del archivo con el que se va a guardar, y otra vez de forma similar a lo hecho en el ejercicio 2.c se cargaran los archivos de forma segura, y una vez cargados, el servidor se encargará de guardarlos en su filesystem.

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

Un servidor con estado es uno tal que mantiene los datos del cliente entre requests, por lo que el cliente puede enviar menos datos entre requests. Esto es muy útil en aplicaciones que requieran logueo, el cliente no tendrá que volver a enviar sus datos cada vez que haga un request. El problema que surge con este tipo de servidores es que pueden existir muchas sesiones y transacciones incompletas, las cuales deberán ser finalizadas de forma inteligente para poder tomar nuevas.
Un servidor sin estado se evita este problema ya que como su nombre lo indica, no guarda ningún estado, por ejemplo, el cliente deberá especificar el nombre de archivo que quiera acceder, la localización de lectura y escritura, y autenticarse en cada request. Las aplicaciones REST son otro gran ejemplo de servidores sin estado
Un servidor sin estado no considera inútil al cliente, y depositara el estado en la caché de este. Otra ventaja de los servidores sin estado es que no requieren la sincronización de los

servidores con estado ya que no necesitan establecer un estado mutuo entre el servidor y el cliente, la respuesta por parte del servidor al request ya tendrá los datos pedidos.