

TP 1: Conceptos básicos para la construcción de Sistemas distribuidos

Sistemas Distribuidos y Programación Paralela Alumno: Esteban Nicolás Larena Legajo: 154582 Profesores: Juan Martín Rodríguez Caillava y David Marcelo Petrocelli



# Punto 1

Consigna: “Escriba un servidor que, usando sockets, reciba un mensaje de texto y lo repita a su cliente. Programe también el cliente para verificar y probar el comportamiento del servidor. Realice la implementación en TCP y comente los resultados”

Para probar este programa se puede ejecutar el siguiente comando.

docker run --name sdypp-tp1-1-server -p 12000:12000 nilare/sdypp-tp1-1-server:latest

Luego, desde una terminal usar Telnet para conectarse al Puerto 12000 de la máquina. Se pueden seguir las instrucciones que le va dando el servidor.

Los registros se guardan en /usr/src/logs/log.log.

# Punto 2

Consigna: Modifique el programa anterior para que pueda atender varios clientes a la vez.

Similar al anterior, se puede probar desde varias terminales.

docker run --name sdypp-tp1-2-server -p 12000:12000 nilare/sdypp-tp1-2-server:latest

Los registros se guardan en /usr/src/logs/log.log.

# Punto 3

Consigna: *Escriba un servidor de mensajes en colas, que permita a los clientes dejar un mensaje (identificando de alguna forma a quién se lo dejan), y bajar los mensajes que le están dirigidos. La comunicación entre cliente y servidor debe ser mediante sockets, y el servidor debe poder atender varios clientes a la vez.*

Para comenzar, hay que ejecutar el proyecto desde el método “main” de la clase “ServerTest”. Unos instantes luego, ya se puede correr el mismo proyecto pero desde el método “main” de la clase “ClientTest”. Para este punto, el siguiente y los anteriores, no programé ninguna acción que pudiese tomar el usuario desde el lado del servidor.

Los registros se guardan en “log.Punto3.log”.

Dicho esto, desde el cliente se le va a pedir que ingrese una opción numeradas del 1 al 3. Se debe ingresar por teclado alguna de ellas y luego presionar “enter” para continuar. También, si se ingresa más de un carácter, sólo se tomará en cuenta el primero, por motivos de diseño.

## Opción 1

Se le pide que ingrese un destinatario y luego un remitente. Estos valores no pueden contener una línea vacía por lo que para continuar ingrese algún valor.

Luego, se le pedirá que ingrese un mensaje, puede contener cualquier valor que desee.

Finalmente, el cliente enviará al servidor un String cuyo contenido tiene formato JSON, o por lo menos se puede parsear con herramientas de este tipo.

Desde el lado del servidor, se devolverá un mensaje sin formato indicando si el mensaje fue archivado o no.

## Opción 2

Para leer mensajes, hay que ingresar el nombre del remitente de los mensajes. Y luego, el programa del cliente enviará una petición que el servidor contestará con un mensaje con formato JSON. Luego de ser parseado por la clase Parser, se muestra el mensaje, con emisor, destinatario, identificador y contenido.

## Opción 3

Simplemente termina el programa.

# Punto 4

Consigna: *Modifique el programa anterior para que el mensaje de la cola sea borrado por el servidor una vez que el cliente confirma, mediante un mensaje de tipo ack, que efectivamente recibió el mensaje que estaba en la cola.*

Funciona igual que el proyecto de la consigna anterior, sólo que se agregó una opción 4: al ingresar a ella, se le va a pedir que ingrese un número entero, el cual representa inequívocamente a un mensaje. Por ello, recomiendo primero recuperar una lista de mensajes e intentar borrar alguno de los mensajes luego, para tener el id a mano.

Los registros se guardan en “logs/Punto4.log”.

# Punto 5

Consigna: *Escribir un servicio que devuelva información de clima del lugar donde reside el servidor. Esta información podrá generarse de forma aleatoria. Deberá ser realizado con RMI. Para ello considere la interface remota, la clase (lado servidor) que implementa esa interface, el servidor, y un cliente que permita probar este funcionamiento.*

Para comenzar ejecute el servidor desde el método “main” de la clase “ServerTest”. Luego, si desea puede ejecutar uno o más clientes desde el método “main” de la clase “ClientTest”.

Los clientes escucharán al servidor constantemente y harán eco de lo q le envíe.

Desde el servidor, puede presionar “enter”. Luego de esto se le pedirá que ingrese la nueva /temperatura.

Los registros se almacenan en “logs/Punto5.log”.

# Punto 6

Consigna: *Escribir un servidor utilizando RMI, que ofrezca la posibilidad de sumar y restar vectores de enteros. Introduzca un error en su código que modifique los vectores recibidos por parámetro. ¿Qué impacto se genera? Qué conclusión saca sobre la forma de pasaje de parámetros en RMI? Mostrar claramente los valores de los vectores del lado cliente, antes y después de la ejecución de la suma o resta.*

Primero ejecutar el método “main” de la clase “ServerTest”. Luego, ejecutar el método “main” de la clase “ClientTest”.

Desde el cliente, se le va a pedir que seleccione una de dos opciones, para sumar o restar vectores respectivamente. Una vez elegida una opción tendrá que cargar los vectores con números enteros. Luego el servidor hará la operación de suma y resta y desde el cliente se mostrarán los resultados.

Para que se generen algunos errores, ejecutar el método “main” de la clase “ClientFailureTest”, el cual usa una clase especialmente modificada para esto.

Las conclusiones que saqué, fueron que desde el servidor no se puede modificar la memoria de los clientes de forma remota. Por lo que si se quisieran pasar los vectores por referencia, se tendrían que almacenar los vectores en el mismo servidor, y luego clonarlos en la memoria del cliente, lo cual no sería pasaje por referencia pero lo simula.

Los registros se almacenan en “logs/Punto6.log”.

# Punto 7

Consigna: *Implemente un servidor RMI que resuelva tareas genéricas. Para ello tener en cuenta la interface Tarea, que tendrá un método ejecutar (). El objetivo es que desde el cliente se puedan escribir objetos (que implementen la interface Tarea) que hagan un cálculo concreto (calcular un número aleatorio, un primo, el valor de Pi con cierta precisión, etc.), y que esa tarea se pase al servidor RMI, quien hará el cálculo y devolverá el valor al cliente.*

Al igual que los anteriores proyectos, se debe empezar ejecutando el método “main” de la clase “ServerTest” y luego el “main” de la clase “ClientTest”.

Desde el lado del cliente se le pedirá que seleccione algún servicio, luego tendrá que ingresar un parámetro. Finalmente, el cliente mostrará el resultado de la operación seleccionada.

Los registros se almacenan en “logs/Punto7.log”.