PRÁCTICA 2: DISEÑO E IMPLEMENTACIÓN DE UN ESQUEMA EDITOR/SUSCRIPTOR (PARTE2)

SISTEMAS DISTRIBUIDOS

Curso 2015-2016

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

JAVIER PRIETO CEPEDA: **100 307 011**

PEDRO ROBLEDO QUINTERO: **100 282 657**

ÍNDICE

TABLAS DE PRUEBAS

1 Introducción

En este documento se procede a la explicación de práctica 2, "Diseño e implementación de un esquema editor/suscriptor (parte 2)" de la asignatura Sistemas Distribuidos del grado en Ingeniería Informática de la Universidad Carlos III de Madrid. La realización de la misma la han llevado a cabo los alumnos Javier Prieto Cepeda y Pedro Robledo Quintero.

En primer lugar, se realizará una breve explicación de las implementaciones incluidas en esta nueva versión de la práctica.

En segundo lugar, se mencionarán las decisiones de diseño tomadas en la implementación.

Posteriormente, se expondrá la batería de pruebas llevada a cabo para la verificación de la aplicación.

Por último, se detalla la forma de compilar y obtener los distintos ejecutables de la aplicación.

2 Descripción del código

En este apartado, se va a proceder a explicar la implementación realizada de la aplicación. Se explicarán por separado, los distintos componentes implementados que conforman la nueva versión.

2.1 Servicio de almacenamiento de textos

Este componente, se incorpora a esta versión de la práctica. Su funcionalidad, consiste en almacenar como máximo los últimos 10 mensajes de cada tema, de forma que pueda enviárselos al intermediario en caso de que se los pida. Se ha decidido por simplicidad, que el servidor de textos almacene los temas en una lista enlazada, de forma que los textos de cada tema sean un array de 10 posiciones indexado en forma de *buffer circular*.

De esta forma, se ha implementado el fichero *"almacenamientoTemas.x"*. En este fichero, se encuentra la interfaz para la generación automática de los ficheros cliente y servidor rpc para el almacenamiento de textos. Consta de las 3 funciones que pueden ser invocadas de forma externa por los clientes: init(), putpair() y gettexts(). Además, incorpora la estructura que devolverá la función *gettexts*.

La función *init*, se encarga de que el servidor de textos inicialice las estructuras y variables para el control de concurrencia necesarias para la gestión de los textos.

La función *putpair*, recibe un tema y un texto asociado a dicho tema. En primer lugar, comprueba que el servicio de almacenamiento de textos ha sido inicializado previamente para poder añadir el nuevo texto. En caso de haber sido inicializado, recorre la lista de temas en busca del tema del nuevo texto. En caso de existir, añadirá el nuevo texto en el array de textos del tema en la posición correspondiente (se calcula aplicando módulo 10). En el caso de que el tema no exista, se creará un nuevo nodo en la lista de temas, y el texto ocupará la primera posición del array de textos. En caso de haber sido realizada de forma satisfactoria, retornará 0. En caso de haberse producido algún error, retornará -1.

La función *gettexts*, recibe el tema del cual se quieren recibir los últimos 10 textos recibidos. En primer lugar, comprueba que el servicio de almacenamiento de textos ha sido inicializado previamente para poder buscar el tema. En caso de haber sido

inicializado, se procede a buscar el tema. Si el tema existe, retornará una estructura. Esta estructura, cuenta con un entero, que indica el número de textos que se devuelven, y un array de char, en el que se incluyen los textos del tema. Este array, tendrá un tamaño 10249 bytes. El tamaño de este array es el resultado de:

$$(num_{Textos} * tam_{Textos}) + num_{\ Separadores}$$

2.2 Broker

El intermediario, incorpora dos nuevas funcionalidades con respecto a la primera práctica. Estas funcionalidades, consisten en la comunicación con el servidor rpc de almacenamiento de textos.

En primer lugar, cada vez que se recibe un nuevo texto generado por un editor, antes de que sea enviado a los suscriptores del tema asociado del texto, se realiza una llamada a la función *putpair* del servidor de almacenamiento de textos, enviando el nuevo texto como argumento.

En segundo lugar, cada vez que se recibe una petición de *subscribe* por parte de un suscriptor, tras finalizar la comunicación con el suscriptor, se realiza una llamada a la función *gettexts* del servidor de almacenamiento de textos, enviando como argumento un puntero a la estructura que en la que nos devolverá la función el resultado. Una vez obtenemos el resultado, en primer lugar comprobamos que existen textos para el tema deseado. En caso de existir textos, se realizan tantas conexiones de envíos de textos al hilo escuchador del suscriptor como textos se han recibido, siguiendo el mismo protocolo que cuando se difunde un nuevo texto.

Además, como novedad, se obtiene por parámetro como último argumento al lanzar el broker, la dirección ip del servidor de almacenamiento de textos.

2.3 Servicio web de transformación de textos

Este componente, se añade a esta nueva versión de la práctica. Es un web-service de java (JAX-WS), y se encargará de recibir un string, convirtiendo en mayúsculas la primera letra de cada una de las palabras que incluye. Para ello, en primer lugar, se prepara la cadena para poder realizar la funcionalidad de forma correcta, de modo que se eliminan los caracteres vacíos (como espacios y tabulaciones) que se encuentran al principio y final de la cadena. Posteriormente, en caso de existir 2 caracteres "espacio" de forma consecutiva, se

reducen a un único carácter. De esta forma, realizamos un "Split" por caracteres "espacio", obteniendo en forma de array cada una de las palabras de la cadena. Una vez obtenido este array, se convierte a mayúscula el primer carácter de cada una de las palabras que lo conforman, y se concatena cada palabra en un nuevo string con un carácter "espacio" entre cada una de ellas. Una vez convertido todo el mensaje, se devuelve el nuevo string con el mensaje convertido.

Por un lado, esta funcionalidad, se implementa en el servicio, que es invocado por el Publisher, que es quien posee la interfaz para las llamadas al servicio web.

2.4 Suscriptor

El suscriptor, incorpora una nueva funcionalidad, que será convertir la primera letra de cada palabra de los mensajes asociados a los temas suscritos a mayúscula.

Para ello, realiza la invocación del servicio web de transformación de textos, enviando cada vez que recibe un nuevo mensaje de los temas suscritos el texto asociado al mensaje a este servicio web, y obteniendo el texto convertido.

Además, para poder realizar esta invocación, se ha modificado la recepción de los argumentos al ejecutar el suscriptor. De esta forma, se ha modificado el método *parseArguments* del suscriptor (proporcionado en el código base) para que pueda recibir un nuevo parámetro, el de la dirección del servicio web de transformación de textos.

3 Decisiones de diseño

En este apartado, se procede a explicar las decisiones de diseño adoptadas para la implementación de esta nueva versión.

- Los textos asociados devueltos por el servidor de almacenamiento de textos, son devueltos en una misma cadena. De esta forma, todo se realiza en una única llamada y es más simple.
- Los textos devueltos en una misma cadena por el servidor de almacenamiento de textos, usan como separador entre sí el carácter ':', puesto que es el único que no está permitido dentro de los textos, y nos sirve para poder separar posteriormente en el broker los textos.

- Por simplicidad (y puesto que no se especifica en el enunciado), una vez recibidos todos los textos por parte del servidor de almacenamiento de textos, estos son enviados al suscriptor en el orden recibido, por el servidor de almacenamiento de textos.
- Por simplicidad, se eliminan dobles espacios y espacios al principio y final de la cadena de texto en el servicio web de transformación de textos, de forma que las palabras puedan ser identificadas con facilidad.

4 Batería de pruebas

En este apartado, se exponen las pruebas realizadas para la verificación de la correcta implementación de la práctica. Se obvian las pruebas realizadas en la primera entrega, y que verificaban la correcta implementación de la misma. Estas pruebas, verifican el correcto funcionamiento de las nuevas funcionalidades.

Obtener textos de tema no registrado	
Objetivo	Comprobar que en caso de no existir un tema en el servidor de textos, no se recibe ningún texto y el funcionamiento es el esperado.
Entradas	Suscriptor: SUBSCRIBE de tema del que no se han recibido mensajes.
Salida	
Conclusión	La salida es la esperada, puesto que al no haber enviado ningún mensaje de ese tema, no hay ningún texto asociado, y por tanto, no se recibe ningún mensaje anterior.

Tabla 1: Textos de tema no registrado

Obtener textos de tema con menos de 10 textos	
Objetivo	Comprobar que para un tema con menos de 10 textos enviados al servidor de textos, éste devuelve correctamente los textos y son enviados al suscriptor con normalidad. Se han enviado previamente 5 textos del tema deportes.
Entradas	Suscriptor: SUBSCRIBE deportes
Salida	Se reciben los 5 mensajes previos de deportes
Conclusión	La salida es la esperada, puesto que se reciben los 5 mensajes previos a la suscripción enviados del tema deportes.

Tabla 2: Tema con menos de 10 textos

Obtener textos de tema con más de 10 textos y tamaño máximo por cada texto y nombre de tema.	
Objetivo	Comprobar que para un tema con más de 10 textos enviados al servidor de textos, éste devuelve correctamente los últimos 10 textos enviados al tema. Además, cada texto se envía con el tamaño máximo (1024 caracteres), de forma que se compruebe el correcto almacenamiento máximo. Además, el nombre del tema, será del tamaño máximo por tema (128 bytes).
Entradas	Suscriptor: SUBSCRIBE ab…bc (tema con 128 bytes).
Salida	Se reciben los últimos 10 mensajes, con una longitud máxima de 1024 bytes por texto.
Conclusión	La salida es la esperada, puesto que se reciben los últimos 10 mensajes sin perder ningún byte.

Tabla 3: Máximo número de textos, tamaño de texto y de tema

Web Service: Obtener texto transformado	
Objetivo	Comprobar que para el texto enviado, se realiza la transformación de la cadena de forma correcto, mandando un texto con espacios consecutivos, y al principio y final del texto.
Entradas	" hola adios "
Salida	"Hola Adios"
Conclusión	La salida es la esperada, puesto que se han eliminado los espacios, y se ha convertido la primera letra de cada palabra a mayúscula.

Tabla 4: Transformación texto

Web Service: Transformar texto en blanco	
Objetivo	Comprobar que para el texto enviado, se realiza la transformación de la cadena de forma correcto, mandando un texto con espacios consecutivos, y al principio y final del texto.
Entradas	" hola adios "
Salida	"Hola Adios"
Conclusión	La salida es la esperada, puesto que se han eliminado los espacios, y se ha convertido la primera letra de cada palabra a mayúscula.

Tabla 5: Transformar texto en blanco

5 Compilación

Para la compilación de la práctica, hemos añadido al MAKEFILE de la entrega anterior, algunas opciones para la compilación de los ficheros nuevos.

Aun así, no están todos añadidos, y hay que seguir la siguiente estructura para la compilación:

```
rpcgen –a –N –M almacenamientoTemas.x
make –f Makefile.almacenamientoTemas
javac transform/transformService.java
javac transform/transformPublisher.java
make
```

De esta forma, en primer lugar, generamos los ficheros necesarios para el servidor de rpc. Posteriormente, se compilan estos ficheros. Por último, se ejecuta el MAKEFILE propio, que compila los ficheros implementados por nuestra parte y generan los ejecutables de la práctica. La traza de compilación, por tanto, es:

```
rpcgen –a –N –M almacenamientoTemas.x
make –f Makefile.almacenamientoTemas

cc   -g     -D_REENTRANT     -c   -o   almacenamientoTemas_clnt.o
almacenamientoTemas_clnt.c
cc   -g     -D_REENTRANT     -c   -o   almacenamientoTemas_client.o
almacenamientoTemas_client.c
cc   -g     -D_REENTRANT     -c   -o   almacenamientoTemas_xdr.o
almacenamientoTemas_xdr.c
cc   -g     -D_REENTRANT         -o   almacenamientoTemas_client
almacenamientoTemas_clnt.o          almacenamientoTemas_client.o
almacenamientoTemas_xdr.o -lnsl -lpthread
cc   -g     -D_REENTRANT     -c   -o   almacenamientoTemas_svc.o
almacenamientoTemas_svc.c
cc   -g     -D_REENTRANT     -c   -o   almacenamientoTemas_server.o
almacenamientoTemas_server.c
cc   -g     -D_REENTRANT         -o   almacenamientoTemas_server
almacenamientoTemas_svc.o           almacenamientoTemas_server.o
almacenamientoTemas_xdr.o -lnsl –lpthread

make

gcc -Wall  -g -Wall  -g -c broker.c
gcc -Wall  -g -Wall  -g -c read_line.c
gcc -Wall  -g -Wall  -g -c database.c
gcc      -L/lib/      broker.o      read_line.o      database.o
```

```
almacenamientoTemas_clnt.o  almacenamientoTemas_xdr.c  -lpthread
-o broker
gcc -Wall  -g -Wall  -g -c publisher.c
gcc -L/lib/ publisher.o read_line.o -lpthread -o publisher
javac -g suscriptor.java
gcc -c serverAlmacenamiento.c
gcc     -o     serverAlmacenamiento     serverAlmacenamiento.o
almacenamientoTemas_svc.o almacenamientoTemas_xdr.c
```

El servidor de almacenamiento de textos, se tiene como nombre "serverAlmacenamiento.c", y su ejecutable es "serverAlmacenamiento". Se ha decidido de ésta forma, para no tener confusiones con el que genera automáticamente el comando rpcgen.

5.1 Otros aspectos

Para el web-Service de JAX-WS, se ha seguido el paso a paso explicado en clase, con la diferencia de modificar los nombres a los paquetes y ficheros. La generación de estos servicios, se han realizado de la siguiente forma:

1. Definición del servicio : transformService.java
2. Clase que publica el servicio: transformPublisher.java
3. Compilación y despliegue:
    a. *javac transform/transformService.java*
    b. *javac transform/transformPublisher.java*
4. Obtención del WSDL:
   wsgen -cp . –wsdl transform.transformService
5. Generación del cliente:
   wsimport -p client -keep transformService.wsdl
6. En nuestro caso, el código del cliente, queda implementado en suscriptor.java, pero tenemos que hacer los "import" necesarios para el correcto funcionamiento.

5.2 Ejecución

Para la ejecución, se han de utilizar los siguientes comandos:

```
# Servidor de textos
./serverAlmacenamiento

#broker
./broker <puertoE> <puertoS> <ipServerAlmacenamiento>

#publisher
./publisher <tema> <ipBroker> <puertoBroker>
```

```
#servicioWeb
java transform.transformPublisher

#suscriptor
java –cp . suscriptor –s <ipBroker> -p <puertoBrokerSusc> -u
<urlWebService>
```

6 Conclusiones

Ha sido interesante la realización de ésta práctica, puesto que resulta de gran ayuda la generación automática de la comunicación distribuida mediante rpc's de Sun y mediante los web-Service de java.

Pero por el contrario, en el caso de las rpc de Sun, hemos encontrado problemas a la hora de comprender la liberación de memoria hacia atrás. Además, hemos tenido problemas a la hora de conseguir un correcto funcionamiento, puesto que el *destroy* del cliente, debe realizarse cuando se finalice la aplicación, y no tras cada invocación de un procedimiento remoto, puesto que si cada vez que se invoca a un procedimiento remoto, se crea y se destruye la estructura del cliente para la llamada, se obtienen muchos errores.

En el caso de los web-service de Java, ha sido interesante ver lo sencillo que resulta la invocación de un servicio web a comparación de las rpc's de Sun.

Por último, destacar que una vez implementada toda la infraestructura de comunicación entre los distintos componentes de forma distribuida, resulta muy gratificante, y ha servido para comprender la complejidad que conlleva la realización de un sistema como el implementado en la práctica.