
Práctica: Sistema Distribuido para Compartición de Archivos

Leganés
Ingeniería Informática
Sistemas Distribuidos

uc3m

Universidad
Carlos III
de Madrid

Laura Sánchez Cerro NIA 100383419 Gr. 81

100383419@alumnos.uc3m.es

Alejandro Prieto Macías NIA 100383428 Gr. 81

100383428@alumnos.uc3m.es

Índice

1. Introducción	2
2. Modelo	3
2.1. Cliente	3
2.2. Servidor	3
2.3. Base de Datos	4
2.3.1. Separadores	4
2.3.2. Ejemplo	4
2.4. Funcionamiento	4
2.5. RPC	5
2.5.1. Cliente RPC	5
2.5.2. Servidor RPC	5
2.6. Servicio Web	5
2.6.1. Cliente	6
2.6.2. Servidor	6
3. Proceso de obtención del código	7
3.1. Compilación	7
3.1.1. Cliente Java	7
3.1.2. Servidor en C	7
3.1.3. Servicio RPC	7
3.1.4. Servicio Web	7
3.2. Ejecución	7
3.2.1. Cliente Java	8
3.2.2. Servidor en C	8
3.2.3. Servicio RPC	8
3.2.4. Servicio Web	8
4. Pruebas	9
4.1. Evidencia de las pruebas	11
5. Conclusiones	12

Introducción

Esta práctica consiste en realizar una aplicación distribuida que se encarga de compartir ficheros entre usuarios registrados en el sistema. Para ello, se apoya en la arquitectura *P2P* para la compartición de los archivos y el control del registro de los usuarios se realiza de forma centralizada en un servidor, aunque el intercambio de archivos se realiza entre los clientes. Además, esta práctica se complementa mediante el uso de *Servicios RPC* con el objetivo de que el almacenamiento de la información lo realice este nuevo servidor. Por último, se añaden un *Servicio Web* para transformar las descripciones de los ficheros a mayúsculas.

Modelo

Cliente

El cliente puede comunicarse con el servidor para solicitar diferentes operaciones:

- Registrarse en el sistema con la sintaxis *REGISTER* <userName>.
- Darse de baja del sistema con la sintaxis *UNREGISTER* <userName>.
- Conectarse en el sistema si el usuario previamente está registrado *CONNECT* <userName>.
- Desconectarse del sistema *DISCONNECT* <userName>.
- Publicar contenido si el usuario está conectado *PUBLISH* <fileName> <description>.
- Eliminar contenido publicado si el usuario está conectado *DELETE* <fileName>.
- Listar los usuarios conectados en el sistema *LIST_USERS*.
- Listar los ficheros publicados por otros usuarios *LIST_CONTENT*.
- Salir de programa mediante *QUIT*. Además, si el usuario se encontraba conectado, se desconectará automáticamente para evitar problemas de consistencia en la Base de Datos.

Un usuario puede comunicarse con otro usuario para conseguir un fichero determinado y copiar su contenido en un nuevo fichero con el nombre que especifique *GETFILE* <userName> <remote_file_name> <local_file_name>. Esta operación conecta primero con el servidor para que le devuelva la lista de usuarios conectados en el sistema con el fin de obtener la dirección IP y puerto del usuario propietario del fichero. Una vez se obtiene el listado, se filtra para conseguir el usuario con el que queramos comunicarnos (**userName**). El fichero buscado (<remote_file_name>) se copia en el nuevo fichero del usuario que solicita la operación (<local_file_name>).

Servidor

El servidor recibirá las peticiones del cliente de las diferentes operaciones mencionadas anteriormente. Para ello, comprobará que la funcionalidad se realiza correctamente y, posteriormente, se devolverá al cliente un código indicando si ocurrió algún error o funcionó correctamente. Se encargará de recibir:

- **REGISTER:** Se encargará de llamar a la función que realiza el registro en la base de datos.
- **UNREGISTER:** Se encargará de llamar a la función que realiza la eliminación del usuario de la base de datos.
- **CONNECT:** Se encargará de llamar a la función que añade los datos de conexión de un usuario en la base de datos.
- **DISCONNECT:** Se encargará de llamar a la función que elimina los datos de conexión de un usuario en la base de datos.
- **PUBLISH:** Se encargará de llamar a la función que añade los datos de un nuevo fichero asociado a un usuario en la base de datos.
- **DELETE:** Se encargará de llamar a la función que elimina los datos de un nuevo fichero asociado a un usuario en la base de datos.
- **LIST_USERS:** Se encargará de llamar a la función que saca la lista de usuarios conectados de la base de datos y la envía al usuario.
- **LIST_CONTENT:** Se encargará de llamar a la función que saca la lista de contenido o ficheros asociado a un usuario de la base de datos y la envía al usuario.

Base de Datos

Para guardar la información de los usuarios registrados, así como los ficheros publicados y su descripción se ha decidido, desde un primer momento, elegir un sistema persistente en disco como puede ser un fichero de texto sobre el que se escriben los datos necesarios. En la primera parte de la práctica era el servidor quien escribía en el fichero y para la segunda parte es el **servidor RPC** quien lo hace. Debido a que hemos elegido realizarlo sobre un fichero de texto, ha sido necesario establecer unos separadores para identificar cada tipo de dato. A continuación, se muestra cómo se ha definido cada dato almacenado. Cabe destacar que si el usuario está conectado aparece guardada la dirección **IP** y el puerto. En caso de estar desconectado, no se mostrará. Además, aparecen los ficheros publicados junto con su descripción:

Separadores

- **Usuario:** :::
- **IP:** \$
- **Puerto:** :
- **Nombre de archivo:** ->
- **Descripción de archivo:** ||

Para evitar mezclar los contenidos, se han restringido los nombres permitidos. Los nombre de usuario y de ficheros no pueden comenzar por los caracteres separadores: :::, ->, \$ para evitar posibles errores.

Ejemplo

```
:::userName
$10.0.0.1:2500
->file || description of the file
->file2 || description
:::userName2
->file || description of the file
```

Funcionamiento

- **REGISTER:** Se busca en el fichero que no exista un usuario con el mismo nombre antes de dar como válido el registro y escribirlo en el fichero.
- **UNREGISTER:** Se busca en el fichero que exista el usuario que se desea eliminar y, si se encuentra, se elimina finalmente.
- **CONNECT:** Se busca en el fichero que exista el usuario y, si se encuentra, se verifica si en la siguiente línea está escrita una dirección IP y puerto. Si no existe significa que el usuario no estaba conectado y, por tanto, se procederá a conectarlo, lo que implica añadir esta línea.
- **DISCONNECT:** Se busca en el fichero que exista el usuario y, si se encuentra, se verifica si en la siguiente línea está escrita una IP y puerto. Si existe significa que el usuario estaba conectado y por tanto se procederá a eliminar esa línea para desconectar al usuario.
- **PUBLISH:** Se busca en el fichero que exista el usuario y se verifica si se encuentra conectado por los medios mencionados anteriormente. Si es así, irá a la línea anterior al siguiente usuario (en el caso de que hubiera) o al final del fichero (en el caso de que sea el último usuario), e inscribirá los datos de nombre de fichero y descripción con el separador adecuado.

- **DELETE:** Se busca en el fichero que exista el usuario y se verifica si se encuentra conectado por los medios mencionados anteriormente. Si es así, buscará en la zona de ficheros del usuario, delimitada por la línea de ip-puerto y por la línea correspondiente al siguiente usuario (o si no hay más usuarios) el fichero que se desea eliminar. Si encuentra el fichero deseado, se eliminará.
- **LIST_USERS:** Se busca cada usuario que esté conectado, es decir, que tenga asociado una IP y devuelve el primero, que se añade a la lista que es la que se devuelve. La lista completa se consigue mediante varias llamadas.
- **LIST_CONTENT:** Se localiza al usuario solicitado y, en caso de encontrarlo, se busca en su zona de ficheros, devolviendo el primer fichero que se añade a la lista. Para ello, se va reduciendo la zona de búsqueda hasta terminar de obtener todos los ficheros. La lista completa se consigue mediante varias llamadas.

RPC

Para incluir esta tecnología es necesario definir primero una interfaz en un lenguaje específico, que comparte muchas similitudes con C. El objetivo de la RPC es externalizar el manejo del fichero que representa la *base de datos* y que se realice mediante un método remoto. En la primera parte, todas las funciones descritas las realizaba el servidor, sin embargo, ahora adquiere un rol de cliente respecto al servidor RPC y será este quien tome el control de las operaciones que se realizan sobre el fichero, devolviendo los datos al servidor en forma de listas.

Para ello, se deben declarar las funciones en la interfaz. Posteriormente, se compilará esta interfaz para crear los archivos necesarios para la comunicación del cliente y servidor. Por tanto, solo queda implementar el **cliente RPC** en el servidor y el **servidor RPC** en el archivo que se ha generado para ello.

Cliente RPC

Para realizar la parte RPC, el servidor se convierte en **cliente RPC** ya que se encarga de llamar a los métodos remotos que son los que realmente manejan la *base de datos*. Para realizar esta parte es necesaria iniciar una conexión con el **servidor RPC**, mediante el protocolo TCP para no perder operaciones y, posteriormente, llamar al método remoto. Tras recibir una respuesta, esta conexión se cerrará.

Servidor RPC

El **servidor RPC** es el encargado de implementar las funciones que manejan la *base de datos*. Las funciones que manejan la base de datos, se encuentran en el archivo *imple.c*, por lo que realmente el **servidor RPC** se encargará de invocar estas funciones y esperar su respuesta que será la que envíe al **cliente RPC**. La interfaz del servicio RPC se ha denominado *storage.x*.

Cabe destacar que las funciones de **LIST_USERS** y **LIST_CONTENT** realizan varias llamadas para formar la lista de usuarios y de contenido, respectivamente, que serán enviadas al **cliente RPC**.

Servicio Web

Se realizará mediante esta tecnología una funcionalidad que será accedida como función remota mediante un **servicio web**. Para ello, se debe definir el servicio en *UpperService.java* y, posteriormente, deberá ser publicado mediante *UpperPublisher.java* para que sea accesible a los clientes.

La funcionalidad que debe realizar es transformar todos los caracteres de un string a mayúsculas. Esta función se usará para convertir todas descripciones de los ficheros que se vayan a publicar en mayúscula antes de enviarlas al servidor para registrarlos.

Cliente

Se conectará al servidor cada vez que se quiera realizar un **PUBLISH** para convertir el string de descripción. La descripción que se recibe del **servicio web** será la que finalmente se envíe mediante *sockets* al servidor.

Servidor

Se encargará de realizar la funcionalidad de conversión del string mencionada. Para ello, mediante comunicación basada en protocolos *HTTP*, se realizará la comunicación del servicio en el que se intercambiarán los string de entrada y el string resultado. El servidor es el encargado de computar la funcionalidad.

Proceso de obtención del código

Compilación

Cliente Java

Para la compilación del cliente desarrollado en Java es necesario realizar el siguiente comando:

```
javac cliente.java
```

Como resultado, obtenemos todas las clases desarrolladas en este archivo *.java* y son *cliente*, *myThread* y *Pair*.

Servidor en C

Para la compilación del código del servidor, se hará uso del *makefile* que se encarga de compilar todos los archivos necesarios con sus dependencias.

```
make server
```

Como resultado obtenemos el ejecutable *server*.

Servicio RPC

Para la compilación de los archivos necesarios para el servicio RPC, se hará uso también del *makefile*.

```
make rpcServer
```

Como resultado, obtenemos el ejecutable *rpcServer*.

NOTA: se puede compilar tanto el *server* como el *rpcServer* a la vez usando solamente **make**.

Servicio Web

Para la creación y compilación de los servicios web, serán necesarias las sentencias:

```
javac upper/UpperService.java
javac upper/UpperPublisher.java
```

Con el fin de obtener el archivo *wsdl* y generar el cliente, serán necesarias las siguientes sentencias:

```
wsgen -cp . -wsdl upper.UpperService
wsimport -p client -keep UpperService.wsdl
```

Ejecución

Se debe seguir el orden de ejecución de los programas que se describe a continuación para el correcto funcionamiento. Primero, se debe ejecutar el el servidor (*server*), luego servidor rpc (*rpcServer*), después el servicio web (*UpperPublisher*) y, por último, el cliente (*client*).

Cliente Java

Para la ejecución del cliente desarrollado en Java, es necesario realizar el siguiente comando:

```
java -cp . client -s localhost -p 2500
```

En este caso, se muestra un ejemplo para ejecutar un cliente que usará un servidor alojado en el host *localhost* con el puerto *2500*.

Debido a que se ha desarrollado la parte de Servicio Web, será necesario tener en ejecución el servicio Web para que el funcionamiento sea correcto. Más adelante se detallará cómo activarlo.

Servidor en C

Para la ejecución del servidor, se hará uso del siguiente comando.

```
./server -p 2500 -r localhost
```

En este caso, se trata de un ejemplo para ejecutar el servidor en el puerto *2500* de la máquina que ejecuta y usando haciendo uso del **servicio RPC** que ejecuta en *localhost*, por lo tanto, en la misma máquina. Será necesario que esté en ejecución el servicio RPC para que el servidor funcione correctamente.

Servicio RPC

Para la ejecución del servicio se ejecuta con el comando:

```
./rpcServer
```

Servicio Web

Ejecución del servicio web:

```
java upper.UpperPublisher
```

Pruebas

Para probar el correcto funcionamiento del programa, se han diseñado una serie de pruebas que se muestran en forma de tabla en donde se expone el objetivo de la prueba y los resultados esperados y obtenidos. El tick indica que el programa ha realizado la operación correctamente y la cruz muestra que la operación ha detectado un error.

Se han contemplado las pruebas de forma que engloben las distintas partes de la práctica. Como hay multitud de pruebas, no se muestra una evidencia de cada una de ellas para evitar sobrecargar la memoria; en su lugar, se mostrará a continuación un ejemplo de funcionamiento agrupando varias de ellas, las que consideramos más representativas.

✓: No error ✗: Error

Descripción de la prueba	Esperado	Obtenido
Realizar una operación sin el <i>servidorRPC</i> iniciado	✗	✗
Registrar un usuario	✓	✓
Registrar un usuario sin fichero de base de datos	✓	✓
Registrar un usuario con la base de datos vacía	✓	✓
Registrar un usuario con un nombre inválido	✗	✗
Registrar un usuario con un nombre ya existente	✗	✗
Borrar un usuario	✓	✓
Borrar un usuario sin fichero de base de datos	✗	✗
Borrar un usuario con la base de datos vacía	✗	✗
Borrar un usuario con un nombre inválido	✗	✗
Borrar un usuario con un nombre no existente	✗	✗
Conectar un usuario	✓	✓
Conectar un usuario sin fichero de base de datos	✗	✗
Conectar un usuario con la base de datos vacía	✗	✗
Conectar un usuario con un nombre inválido	✗	✗
Conectar un usuario con un nombre no existente	✗	✗
Desconectar un usuario	✓	✓
Desconectar un usuario sin fichero de base de datos	✗	✗
Desconectar un usuario con la base de datos vacía	✗	✗
Desconectar un usuario con un nombre inválido	✗	✗
Desconectar un usuario con un nombre no existente	✗	✗
Publicar un fichero con descripción	✓	✓

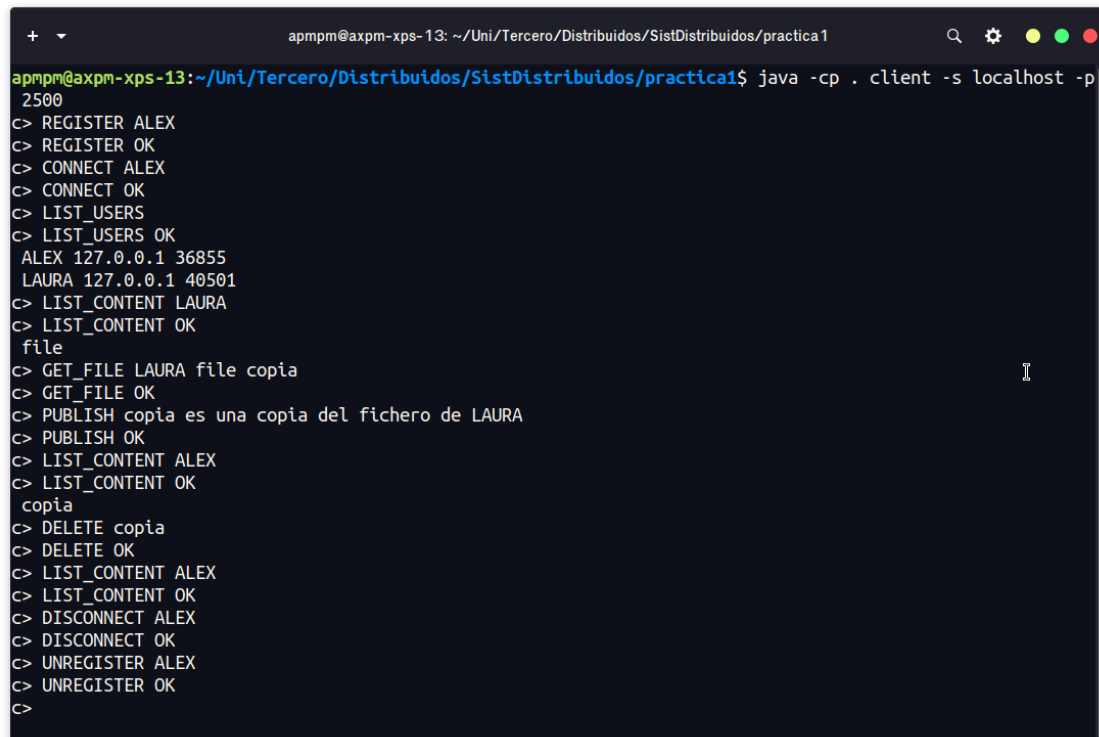
Sigue en la página siguiente.

Descripción de la prueba	Esperado	Obtenido
Publicar un fichero sin descripción	✗	✗
Publicar un fichero sin estar conectado	✗	✗
Publicar un fichero con un nombre inválido de descripción	✗	✗
Publicar un fichero con un nombre inválido del fichero	✗	✗
Publicar un fichero con un nombre de fichero ya existente para el usuario	✗	✗
Publicar un fichero con un nombre de fichero ya existente para otro usuario	✓	✓
Publicar un fichero con una descripción en minúscula y se publique en mayúscula	✓	✓
Publicar un fichero con una descripción en mayúscula y se publique en mayúscula	✓	✓
Eliminar un fichero existente para el usuario	✓	✓
Eliminar un fichero no existente para el usuario	✗	✗
Eliminar un fichero que no existe en el usuario, pero existe en otro usuario	✗	✗
Eliminar un fichero sin estar conectado	✗	✗
Eliminar un fichero con un nombre inválido del fichero	✗	✗
Eliminar un fichero que exista en este usuario y en otro	✓	✓
Listar usuarios sin estar conectado	✗	✗
Listar usuarios estando conectado	✓	✓
Listar contenido sin estar conectado	✗	✗
Listar contenido estando conectado	✓	✓
Listar contenido estando conectado de un usuario no existente	✗	✗
Listar contenido estando conectado de un usuario con un nombre inválido	✗	✗
Obtener archivo	✓	✓
Obtener archivo sin estar conectado	✗	✗
Obtener archivo de un usuario no conectado	✗	✗
Obtener archivo que no pertenece a un usuario o no existe	✗	✗
Obtener archivo de usuario con nombre inválido	✗	✗
Obtener archivo de usuario que no existe	✗	✗
Obtener archivo con nombre de fichero remoto inválido	✗	✗
Obtener archivo con nombre no existente	✗	✗
Obtener archivo con nombre de fichero local inválido	✗	✗

Tabla 1: Batería de Pruebas

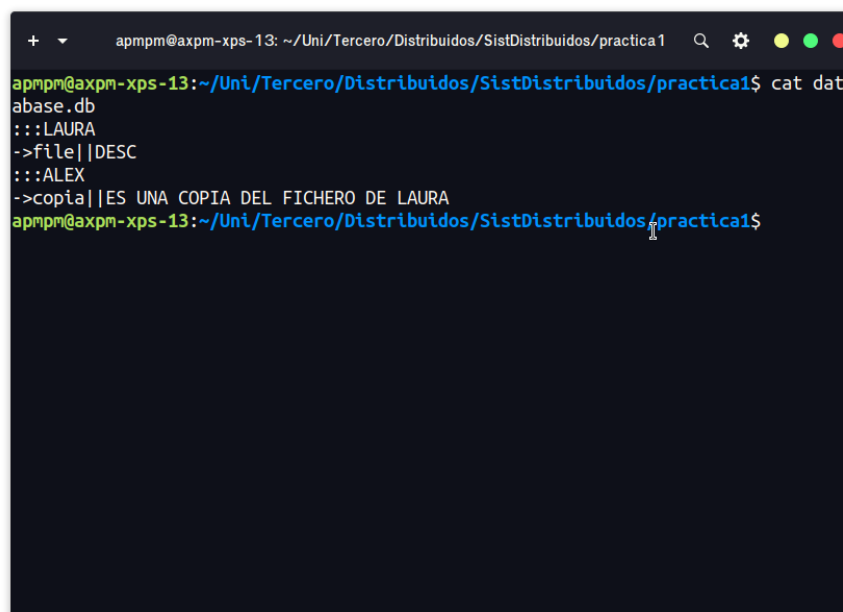
Evidencia de las pruebas

A continuación, se muestra un ejemplo de las pruebas realizadas para mostrar el funcionamiento básico del programa.



```
apmpm@axpm-xps-13: ~/Uni/Tercero/Distribuidos/SistDistribuidos/practica1
apmpm@axpm-xps-13:~/Uni/Tercero/Distribuidos/SistDistribuidos/practica1$ java -cp . client -s localhost -p 2500
<> REGISTER ALEX
<> REGISTER OK
<> CONNECT ALEX
<> CONNECT OK
<> LIST_USERS
<> LIST_USERS OK
ALEX 127.0.0.1 36855
LAURA 127.0.0.1 40501
<> LIST_CONTENT LAURA
<> LIST_CONTENT OK
file
<> GET_FILE LAURA file copia
<> GET_FILE OK
<> PUBLISH copia es una copia del fichero de LAURA
<> PUBLISH OK
<> LIST_CONTENT ALEX
<> LIST_CONTENT OK
copia
<> DELETE copia
<> DELETE OK
<> LIST_CONTENT ALEX
<> LIST_CONTENT OK
<> DISCONNECT ALEX
<> DISCONNECT OK
<> UNREGISTER ALEX
<> UNREGISTER OK
<>
```

Figura 1: Cliente



```
apmpm@axpm-xps-13: ~/Uni/Tercero/Distribuidos/SistDistribuidos/practica1
apmpm@axpm-xps-13:~/Uni/Tercero/Distribuidos/SistDistribuidos/practica1$ cat database.db
:::LAURA
->file||DESC
:::ALEX
->copia||ES UNA COPIA DEL FICHERO DE LAURA
apmpm@axpm-xps-13:~/Uni/Tercero/Distribuidos/SistDistribuidos/practica1$
```

Figura 2: Base datos tras publicar el archivo

Conclusiones

Esta práctica ha resultado muy útil para acercarnos a los sistemas distribuidos de una manera más directa mediante la realización de un programa funcional. Los conocimientos aprendidos y usados durante este proyecto abarcan distintas ramas de los sistemas distribuidos, esto nos ha permitido ser conscientes de las múltiples maneras de enfocar estas arquitecturas. Además, hemos podido probar las ventajas y desventajas que poseen algunas de las implementaciones que son clave a la hora de elegir la tecnología para desarrollar aplicaciones.

Consideramos que esta arquitectura de sistemas distribuidos es muy interesante y, como se sabe, ha creado un gran impacto en los sistemas actuales. Por ello, este pequeño acercamiento cobra mayor importancia.