Programación de Sistemas Distribuidos

Curso 2015/2016

Práctica 1

Diseño e implementación de un **S**istema de **M**ensajería **I**nstantánea utilizando Servicios Web

Índice

1. Introducción

El objetivo de esta práctica es diseñar e implementar un **Servicio de Mensajería Instantánea** (**I**nstant **M**essaging **S**ervice) utilizando Servicios Web. En este caso, se va a utilizar la librería **gSOAP** para realizar la comunicación entre las distintas partes del sistema.

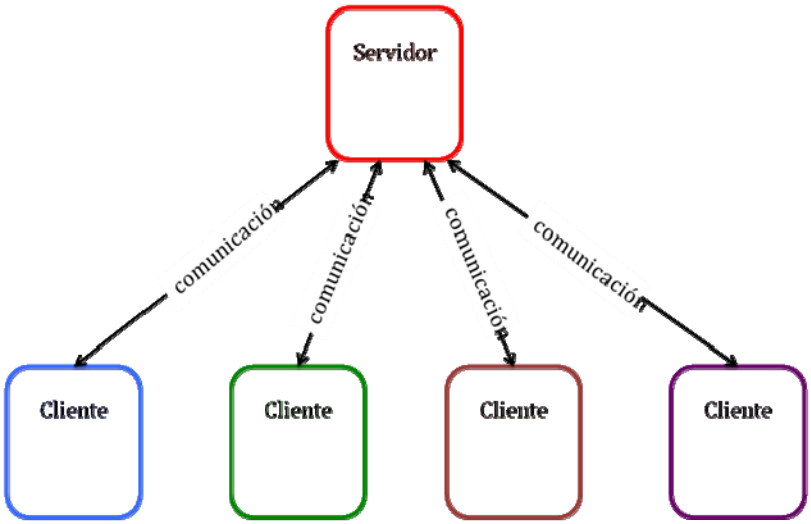
La arquitectura del sistema a desarrollar debe seguir el modelo **cliente-servidor**. De esta forma, el alumno deberá desarrollar las dos partes del sistema: cliente y servidor. La parte cliente será la aplicación utilizada por los usuarios para enviar y recibir los mensajes. La parte servidor será la aplicación que implemente los servicios correspondientes para permitir el intercambio de mensajes entre los clientes.

El lenguaje de programación requerido para desarrollar la práctica íntegramente es **C**. Una vez desarrollada la práctica, ésta se probará sobre el Sistema Operativo **Linux** instalado en el laboratorio donde se han impartido las clases. Por ello se recomienda que, aunque se haya programado la práctica en otro ordenador ajeno al laboratorio, se realicen pruebas en los ordenadores del laboratorio antes de entregarla.

2. Descripción de la arquitectura del sistema

Como se ha comentado anteriormente, el sistema a desarrollar sigue el modelo cliente-servidor. En este caso, dicho sistema debe permitir que varios clientes que usen este servicio (IMS), intercambien mensajes de texto entre ellos.

La comunicación de cada cliente se realizará únicamente con el servidor (ver Ilustración 1). **No se permite la comunicación directa entre dos clientes del sistema.**



3. La parte servidor

La aplicación servidor debe permitir la gestión de varias conexiones por parte de los clientes **de forma concurrente**. Es decir, varios clientes pueden estar conectados al servidor y utilizando los servicios al mismo tiempo.

Asimismo, el servidor deberá contener la funcionalidad correspondiente para la gestión de usuarios. Esto es, por un lado, controlar qué usuarios están dados de alta en el sistema y cuáles no. Únicamente podrán utilizar este sistema los usuarios dados de alta. Por otro lado, únicamente podrán intercambiarse mensajes y ficheros aquellos usuarios que se consideren "amigos", siendo un "amigo" un usuario al que previamente se ha aceptado, mediante una solicitud de amistad, para poder intercambiar con él información. En caso de que el usuario al que va dirigido un mensaje no esté conectado, el servidor deberá almacenar el mensaje para poder disponer de ello cuando se conecte.

La información que mantiene el servidor[1] debe almacenarse de forma persistente (¿cada vez que hay un dato nuevo o periódicamente?), en previsión de caídas del servidor. Una posible prueba puede consistir en reiniciar el servidor. La información que mantiene el cliente puede, o no, almacenarse de forma persistente. En caso que no, el cliente tendrá que pedir información al servidor al arrancar[2].

La aplicación puede, o no, estar diseñada asumiendo que los usuarios se conectan siempre desde el mismo dispositivo; debe documentarse en la memoria la variante que se implementa ("usuario atado" o "usuario móvil"). El sistema debe evitar que el mismo usuario se conecte dos veces simultáneamente y, en el caso del "usuario atado", que el usuario se conecte desde un puesto otro que el en que se dio de alta. En el caso del "usuario móvil", el cliente tendrá que pedir datos al servidor al arrancar. En el caso del "usuario atado" el cliente puede, o no, necesitar pedir datos al servidor al arrancar, dependiendo de cómo está implementado.

3.1 Servicios ofrecidos

La lista de servicios que ofrece el sistema a los usuarios del mismo se detalla a continuación:

Parte obligatoria

1. Dar de alta y dar de baja a un usuario
2. Abrir y cerrar una sesión de un usuario (¡las sesiones deben caducar!)
3. Enviar una solicitud de amistad a otro usuario
4. Enviar un mensaje de texto a otro usuario
5. Aceptar o rechazar la solicitud de amistad de otro usuario

[1] Los mensajes (¿solo los pendientes de ser entregados o todos?), las amistades actuales, las peticiones de amistad pendientes, los avisos de entrega pendientes (y, en caso de permitir el uso de grupos de usuarios, más cosas).

[2] Si el cliente pidiera información al servidor cada vez que la quisiera usar en vez de al arranque, se generaría un número excesivo de mensajes y bajaría mucho el rendimiento de la aplicación.

6. Recibir mensajes de texto de otros usuarios
7. Recibir solicitudes de amistad de otros usuarios
8. Recibir la confirmación de la entrega de mensajes enviados previamente (*double check*)

Parte opcional (un punto extra por cualquiera de ellos pero puntuación max. = 10)

9. Definir y usar grupos de usuarios[3]
10. Adjuntar ficheros a los mensajes
11. Usar SSL

NOTAS:

a) Los servicios del punto 2 solo se implementarán si se elige implementar la variante "usuario móvil".
b) Se puede combinar varios servicios en uno solo (por ejemplo, servicios 6-8).

4. La parte cliente

La parte cliente es la aplicación que utilizarán los usuarios para interactuar con los servicios ofrecidos. Esta aplicación está formada por:

- Interfaz de usuario
- Comunicación con el servidor

La aplicación de usuario se desarrollará en modo texto. De esta forma, el usuario debe ser capaz de poder intercambiar mensajes y ficheros con otros usuarios que estén dados de alta en el sistema. Lo más cómodo es proporcionar una serie de acciones en la interfaz, de tal forma que el usuario indique la acción que desea ejecutar. Se puede ver un ejemplo en la siguiente figura:

```
1) Dar de alta usuario
2) Dar de baja usuario
3) Listado de usuarios amigos
4) Enviar mensaje a un usuario
5) Mostrar nuevos mensajes entrantes
6) …
```

También se permitirá el uso de la aplicación por medio de comandos. En caso de elegir este modo, la interfaz deberá mostrar cada uno de los comandos con todos sus argumentos. Se puede ver un ejemplo en la siguiente figura:

[3] Un posible conjunto (bastante exhaustivo) de servicios remotos para implementar grupos: (a) Crear un grupo; el creador será el administrador del grupo creado (b) Añadir un amigo a un grupo del que soy administrador (c) Borrar un amigo de un grupo del que soy administrador (d) Recibir un aviso de mi adición a, o mi borrado de, un grupo (e) Pedir la membresía de un grupo del que soy miembro (f) Enviar un mensaje a un grupo del que soy miembro; cualquier miembro del grupo que no es amigo mío recibirá una petición de amistad de mi parte de obligada aceptación si quiere recibir el mensaje. Quizás también: (g) Pasar a otro usuario el rol de administrador que desempeño en un grupo y, finalmente, un procedimiento en el servidor: Borrar un grupo que se ha quedado vacío (sin miembros) durante más de un cierto tiempo.

```
registrar usuario (Dar de alta a un usuario)
eliminar usuario (Dar de baja a un usuario)
enviar msg usuarioDst (Enviar el mensaje msg a usuarioDst)
listUsers (Muestra por pantalla la lista de usuarios amigos)
checkInMsg (comprueba los mensajes de entrada)
. . .
```

En todo caso, aunque no se impone un diseño cerrado de la interfaz de usuario, es responsabilidad del alumno permitir que con dicha interfaz se puedan acceder a todos los servicios comentados en la práctica.

La localización del servidor puede estar almacenada en un constante, al igual que en una aplicación como WhatsApp, o puede obtenerse a través de un parámetro de la línea de comandos. En los dos casos, la elección tiene que estar documentada en la memoria. En el primer caso, la parte cliente de la aplicación recibirá un único parámetro de entrada, el nombre (*nick*) del usuario que va a utilizarla. De esta forma, una posible ejecución puede ser:

```
$> client kenny
```

En el segundo caso, la aplicación recibirá dos parámetros de entrada: primero el nombre del usuario y segundo la localización del servidor.

Tanto en el cliente como en el servidor, si el usuario pulsa Ctrl-C se finalizará la ejecución pero, antes de hacerlo, el programa debe llevar a cabo algunas acciones concretas para asegurar la coherencia de datos entre cliente y servidor. En el caso del cliente, no se debe finalizar la ejecución hasta que haya terminado cualquier invocación remota que esté en curso[4]. En el caso del servidor, no se debe finalizar la ejecución hasta que se haya volcado a disco los datos almacenados en memoria. Un cliente también puede querer volcar datos a disco antes de finalizar, sobre todo si se trata de un "cliente atado". Realizar las gestiones pertinentes antes de finalizar implica capturar la señal generada por la introducción del Control-C.[5]

[4] Obsérvese que si el cliente y el servidor están en la misma LAN, en particular, si el cliente y el servidor están localizados en dos ordenadores del mismo laboratorio de la FDI, el tiempo de ejecución de cualquier invocación remota será tan corto que la probabilidad que se manifiesta el problema que se busca resolver aquí es muy baja, pero no sería el caso si el cliente y el servidor estuvieran en LANs distintos.

[5] La manera más fácil de capturar la señal es con la función `sigprocmask()` de la biblioteca GNU. Ver también la sección *How to be a proper program*, `www.cons.org/cracauer/sigint.html`

5. Recomendaciones para desarrollar la práctica

Aunque no es obligatorio, se recomienda que para el desarrollo de la práctica se siga el siguiente orden:

1. Interfaz en el cliente para interactuar con el usuario
2. Altas y bajas de usuarios
3. Solicitudes y aceptaciones de amistad sin persistencia
4. Envío de mensajes sin persistencia
5. *Double check* sin persistencia
6. Persistencia (mensajes, amistades y sus solicitudes, entregas pendientes)
7. Captura de la señal del Control-C
8. Concurrencia en el servidor
9. Envío de mensajes a grupos
10. Envío de ficheros
11. Uso de SSL

NOTA: La parte obligatoria, según este esquema, abarca los puntos del 1 al 8.

6. Entrega de la práctica

La entrega de la práctica debe realizase a través de Campus Virtual. Para ello, se habilitará un enlace donde se podrá subir el material correspondiente. **Todos los ficheros deberán entregarse en formato comprimido (zip) en un único fichero.**

Los ficheros obligatorios para la entrega de la práctica son:

- `client.c`
- `server.c`
- `ims.h`
- `makefile`
- `memoria.pdf`

En caso de añadir nuevos ficheros con código fuente, éstos deberán estar contenidos en el fichero `.zip` entregado. Además, el fichero `Makefile` deberá contemplar los cambios oportunos para compilar de forma correcta, tanto la parte cliente como la parte servidor.

6.1 Requisitos para la entrega de la memoria

Además de entregar los ficheros con el código fuente, se deberá entregar una memoria descriptiva de la práctica. El formato de esta memoria deberá ser **pdf**.

La memoria de la práctica deberá contemplar, al menos, los siguientes aspectos:

- Descripción de cada servicio implementado
- Descripción de la parte cliente
    - Estructuras de datos utilizadas
    - Diseño de la interfaz de usuario
    - Comunicación con el servidor/servicios
- Descripción de la parte servidor
    - Estructuras de datos utilizadas
    - Gestión de usuarios
    - Gestión de mensajes
    - Gestión de ficheros

La fecha límite para la entrega de la práctica es el **16 de Diciembre de 2015**, a las 17:00 horas.

NOTA: **El número máximo permitido de páginas es de 15 (sin contar portada e índice)**, con un tamaño de letra 11 a espacio simple. Se permite la inclusión de figuras, esquemas o capturas de pantalla que faciliten la comprensión de la memoria.

7. Preparar el ordenador (laboratorio)

En los ordenadores del laboratorio ya está instalada una versión de gSOAP (2.8.22). Para poder compilar la práctica, será necesario incluir en el fichero `.bashrc` las siguientes líneas:

```
export GSOAP_HOME=/usr/local/gsoap
export GSOAP_LIB=${GSOAP_HOME}/lib
export GSOAP_INCLUDE=${GSOAP_HOME}/include
PATH=$PATH:$GSOAP_HOME/bin/
```

```
$> source .bashrc
```