uc3m | Universidad Carlos III de Madrid

ESCUELA POLITÉCNICA SUPERIOR

SISTEMAS DISTRIBUIDOS

Práctica Final

Grupo 81

Gabriel García Martínez – 100429061 - 100429061@alumnos.uc3m.es

Jorge Ferrer Hernáez – 100428961 - 100428961@alumnos.uc3m.es

Índice

Descripción del código:

PARTE 1

Introducción

La práctica que hemos realizado consiste en un sistema distribuido que simula a la aplicación *Whatsapp*. Su estructura está basada en una arquitectura cliente-servidor, en la cual una serie de usuarios (**clientes**) se envían y reciben mensajes a través de un **servidor** central.

Los usuarios se almacenan en el servidor en una lista enlazada junto a los mensajes que se quedan en espera para ser recibidos que también se guardan en una lista enlazada.

Cada usuario tiene en su nodo el nombre de usuario, su estado (conectado o desconectado), su dirección ip y puerto para cuando esté conectado, un contador de id de los mensajes para asignar un id a los sucesivos mensajes que envía ese usuario, el id del último mensaje que recibe, otra lista enlazada para guardar los mensajes que se quedan en espera para ser enviados al usuario y un puntero para el siguiente nodo.

En esta lista de mensajes en espera cada nodo tiene el usuario remitente, el mensaje, el id asociado al mensaje y un puntero al siguiente nodo.

Funcionalidades:

El servidor se divide en dos partes: el main y la función tratar petición. En el main se crea el socket TCP que va a utilizar para comunicarse con los clientes. En la función tratar petición están los servicios para cada una de las funciones implementadas en el cliente

Para cada una de las distintas funcionalidades de nuestro programa, abrimos un socket en nuestro cliente, lo conectamos al socket del servidor, realizamos la operación y lo cerramos. El cliente puede ejecutar las siguientes funciones:

Registro
En esta función, registramos en nuestro sistema a un nuevo usuario. Para ello, primero comprobamos que no se introduzca un nombre erróneo: el servidor comprueba que no exista el nombre de usuario introducido por el cliente. Si ese nombre no existe, el registro se realiza con éxito, metiendo al usuario en la lista enlazada e inicializando cada uno de los campos que componen el nodo.

El servidor devolverá el código de error 0 si se ha introducido correctamente, 1 en caso de que ya exista el usuario a introducir y 2 en caso de que haya surgido un error. El sistema mostrará un mensaje correspondiente a cada código de error devuelto.

Baja
En esta función se realiza la baja de un usuario registrado. Hemos supuesto que esta función se realiza cuando el usuario que realiza la baja está desconectado, ya que no lo indicaba el enunciado y podría dar problemas. El cliente envía el nombre del usuario, y el servidor es el que se encarga de comprobar que el usuario se había registrado previamente. En ese caso, el servidor borra al usuario de la lista enlazada de usuarios y devuelve el código de error 0. En el caso de que no exista, devuelve 1, y de que haya ocurrido un error, 2.

Conexión
Esta función es la que se encarga de conectar a los usuarios que previamente se han registrado. El cliente antes de establecer la conexión se encargará de saber si ya existe un usuario conectado o no. Para ello, utilizamos una variable global que indica el nombre del usuario conectado (en caso de que exista). Si ya hay un usuario conectado, se imprimirá el correspondiente código de error: 0 si todo ha ido bien, 1 si el usuario no existe, 2 si ya hay un usuario conectado en esa terminal y 3 si ha habido algún otro error. Si no hay ningún usuario conectado, se seguirá con la función para conectar al usuario.
Si estás intentando conectar a un usuario que no existe y ya hay un usuario conectado, el programa devolverá error de usuario conectado, ya que el sistema hace esta comprobación primero.

Suponemos que para esta práctica conectamos **un usuario distinto por terminal**, es decir, un mismo usuario no se va a conectar desde dos terminales distintas ni en una misma terminal se van a conectar varios usuarios.

Lo siguiente que realiza el cliente es enviar el nombre del usuario que se quiere conectar, y será el servidor el encargado de comprobar si el usuario ya se ha registrado anteriormente buscando en la lista enlazada a ese usuario. En el caso de que no exista, devolverá el código de error correspondiente y el cliente finalizará el procedimiento. En el caso de que el usuario ya estuviese registrado, se enviará la confirmación (código de error a 0) al cliente.

Si el cliente recibe un 0 creará un socket nuevo tipo servidor y un hilo con una función que se encargue de recibir todos los mensajes que el servidor envíe a ese usuario, diferenciando si se tratan de mensajes normales o ACK. Posteriormente se le enviaŕa al servidor la ip y el puerto asociado a ese nuevo socket y será el servidor el encargado de actualizar los campos de ese usuario dentro de la lista enlazada

Esta funcionalidad también cubre los mensajes pendientes: si el usuario está desconectado, y tiene mensajes pendientes de leer en su lista de mensajes, cuando se conecte el servidor se conectará al hilo que recibe mensajes del usuario que se acaba de conectar cada vez que envíe un mensaje y le enviará cada mensaje de su lista (de más antiguo a más reciente) borrándose en el proceso y el hilo del cliente lo imprimirá. Una vez que el usuario

haya recibido los mensajes, esta función también es la que se encarga de enviar los ACK de estos mensajes. El servidor comprueba que el usuario remitente está conectado. De ser así le envía directamente el ack al remitente, de lo contrario el servidor mete un mensaje especial sin remitente que equivale al mensaje ack en la lista de espera del remitente. Por lo que, cuando el remitente se vuelva a conectar se diferenciará si es un mensaje ack o un mensaje normal y con el código de operación correspondiente se le enviarán todos los mensajes de su lista de mensajes en espera como ya se ha indicado anteriormente

Desconexión
Esta función se encarga de desconectar a un cliente. El cliente comprueba si el usuario estaba previamente conectado, y de no estarlo devuelve código de error 2. Luego se le envía el nombre del usuario que se quiere desconectar al servidor que comprobará si el usuario a desconectar existe, en caso contrario devolverá código de error 1. Si ha habido algún otro error en la conexión, 3. Si el usuario existe el servidor devolverá 0 como código de error y de actualizar los campos de ese usuario en la lista enlazada. El cliente cuando recibe el código de error 0 se encargará de cerrar la conexión del socket de su hilo que recibe mensajes con un shutdown y esperará a que el hilo termine su ejecución.

Envío de mensajes:
Esta parte del sistema se encarga de enviar mensajes de un usuario a otro por medio del servidor.
Lo primero que comprueba el cliente es si se está conectado y que no se intente enviar un mensaje al mismo usuario de esa terminal. Si las dos condiciones anteriores no se cumplen el cliente enviará el remitente, el destinatario y el mensaje y esperará al código de error del servidor. El servidor una vez reciba lo antes mencionado comprobaŕa si ambos usuarios existen (de no ser así, devuelve código de error 1). En caso de otro error, devuelve 2 (remitente no conectado, o error en la conexión). Después el servidor introducirá en la lista de mensajes del usuario destinatario el mensaje con los campos correspondientes actualizando el contador de id de mensajes del usuario. Una vez hecho esto el servidor se comportará diferente dependiendo de si el destinatario está conectado o no:

- **Ambos usuarios están conectados:** Se saca el mensaje de la lista de mensajes y el servidor se lo envía al hilo del destinatario enviando el ACK al hilo del remitente en tiempo real.
- **Destinatario desconectado:** el mensaje queda almacenado en la lista de mensajes del destinatario, quien lo recibirá cuando se conecte.

En ambos casos, cuando el usuario conectado se conecte, se actualizará el contador del último mensaje recibido del destinatario. Además, cuando el destinatario lea el mensaje, el servidor enviará un mensaje de confirmación ACK al cliente remitente, para confirmarle que el destinatario ha recibido el mensaje (este mensaje NO actualizará el contador de último mensaje recibido del remitente).

PARTE 2

Introducción

En esta parte de la práctica vamos a crear un servicio web que formatea los mensajes que va a enviar el cliente. Este servicio tomará el mensaje y eliminará todos los espacios sobrantes: solamente puede haber un espacio entre dos palabras consecutivas por cada mensaje enviado.

Funcionalidad añadida: servicio web

Para el desarrollo de esta parte, hemos utilizado las librerías *spyne* y *zeep*, las cuales te permiten desarrollar un servicio web y alojarlo en una url especificada (por simplicidad del ejercicio pedido en la práctica, lo hemos alojado en el localhost), y conectarte a dicho sitio web desde un cliente.

En el desarrollo de esta función, hemos creado una clase para el servicio web denominada "Espaciador", con un método *conversor* que realiza la funcionalidad indicada en la introducción, mediante un sencillo programa de python. Posteriormente, crea una aplicación introduciendo los parámetros correspondientes para ejecutar esta función y lo ejecuta indefinidamente.

Para la ejecución hemos clonado un proceso en el servidor para que ejecute en background el *web service*, y hemos hecho que el cliente llame a la función de *web client* (conectándose al *web service*) y formatee los mensajes antes de llamar a su propia función de enviar mensaje.

Compilación y ejecución:

Compilación:

Para compilar los ficheros de nuestro ejercicio, hemos utilizado un Makefile en el que incluimos comandos para compilar el archivo del servidor y generar su ejecutable, así como para la compilación de las funciones de ambas listas enlazadas, que se encuentran en los ficheros *nodo.h* y *linked-list.c*.

Ejecución:

Para ejecutar, abriremos *n* terminales para conectar *n* clientes simultáneamente (el sistema es **concurrente**: ejecuta *n* hilos, uno para cada cliente).

El comando a utilizar para ejecutar el servidor es el siguiente:

./servidor -p <port_number>

Para el correcto funcionamiento del servicio web, necesitamos instalar dos librerías, ejecutando los siguientes comandos:

pip install zeep
pip install spyne

Necesitamos a su vez abrir una nueva terminal donde ejecutemos el siguiente comando:

python3 ws-space-service.py

Este comando pondrá en funcionamiento el servicio web, el cual será utilizado por cada cliente gracias a su propia implementación del código.

Y para ejecutar, en cada cliente:
python3 client.py -s <ip_addr> -p <port_number>

Donde <*ip_addr*> indica la dirección del servidor, y <*port_number*> el puerto al que se conecta del servidor.

Para la introducción de los comandos en la terminal de cada cliente, hemos controlado el caso de que la sintaxis de cada comando sea la correcta. Además, es necesario que cada cliente ejecute sus comandos internos después de haber ejecutado el servidor; de otra forma, se producirá un error de conexión en el socket.

Batería de pruebas:

A continuación, desarrollamos una batería de pruebas con la que podemos comprobar que se ejecutan correctamente cada una de las funcionalidades descritas en el apartado anterior:

Registro: comprobamos los siguientes casos: el usuario se registra correctamente, el usuario ya existía, intento registrar más de un usuario a la vez.

Comando	Resultado esperado	Resultado obtenido
REGISTER pepe	REGISTER OK cop 0	REGISTER OK cop 0
REGISTER pepe	USERNAME IN USE cop 1	USERNAME IN USE cop 1
REGISTER jorge pepe	Syntax error. Usage: REGISTER <username>	Syntax error. Usage: REGISTER <username>

Baja: comprobamos los siguientes casos: el usuario se da de baja correctamente, el usuario no existía, intento dar de baja a más de un usuario a la vez.

Comando	**Resultado esperado**	**Resultado obtenido**
UNREGISTER pepe	UNREGISTER OK cop 0	UNREGISTER OK cop 0
UNREGISTER pepe2	USER DOES NOT EXIST cop 1	USER DOES NOT EXIST cop 1
UNREGISTER carlos pepe juan	Syntax error. Usage: UNREGISTER <username>	Syntax error. Usage: UNREGISTER <username>

Conexión: comprobamos los siguientes casos: el usuario se conecta correctamente, el usuario no existía, el usuario ya estaba conectado, intento conectar a un usuario mientras otro está conectado en esa terminal, intento conectar a más de un usuario a la vez.

Comando	**Resultado esperado**	**Resultado obtenido**
CONNECT emilio	CONNECT FAIL, USER DOES NOT EXIST cop 1	USER DOES NOT EXIST cop 1
CONNECT pepe	CONNECT OK cop 0	CONNECT OK cop 0
CONNECT emilio (estando pepe conectado)	USER ALREADY CONNECTED cop 2	USER ALREADY CONNECTED cop 2
CONNECT jorge (jorge está registrado)	USER ALREADY CONNECTED cop 2	USER ALREADY CONNECTED cop 2
CONNECT juan carlos	Syntax Error. Usage: CONNECT <name>	Syntax Error. Usage: CONNECT <name>

Desconexión: comprobamos los siguientes casos: el usuario se desconecta correctamente, el usuario a desconectar no existe, el usuario existía y ya estaba desconectado, intento desconectar a más de un usuario a la vez

Comando	**Resultado esperado**	**Resultado obtenido**
DISCONNECT pepe	DISCONNECT OK cop 0	DISCONNECT OK cop 0
DISCONNECT juan (el usuario juan no existe)	DISCONNECT FAIL / USER DOES NOT EXIST cop 1	DISCONNECT FAIL / USER NOT CONNECTED cop 2
DISCONNECT jorge (jorge está desconectado)	DISCONNECT FAIL / USER NOT CONNECTED cop 2	DISCONNECT FAIL / USER NOT CONNECTED cop 2

DISCONNECT juan carlos	Syntax Error. Usage: DISCONNECT <name>	Syntax Error. Usage: DISCONNECT <name>

En este caso, el flujo del programa no permite comprobar que el usuario no existe sin comprobar antes si está conectado o no. Por esta razón, el resultado obtenido de la segunda prueba difiere del resultado esperado.

Envío de mensajes: comprobamos los siguientes casos: el mensaje se envía correctamente en tiempo real, el mensaje se envía y se almacena en la lista de mensajes del destinatario, el destinatario no existe, el remitente no está conectado, el remitente se envía un mensaje a sí mismo.

Para las siguientes pruebas, suponemos que el usuario *carlos* está conectado:

Comando	**Resultado esperado**	**Resultado obtenido**
SEND pepe hola	SEND OK - MESSAGE <id> ACK OF MESSAGE <id> RECEIVED cop 0	SEND OK - MESSAGE 1 ACK OF MESSAGE 1 RECEIVED cop 0
SEND juan hola juan (el usuario juan existe)	SEND OK - MESSAGE <id> cop 0	SEND OK - MESSAGE 1 cop 0
SEND rodrigo hola	SEND FAIL / USER DOES NOT EXIST cop 1	SEND FAIL / USER DOES NOT EXIST cop 1
SEND pepe hola (carlos se ha desconectado)	SEND FAIL cop 2	SEND FAIL cop 2
SEND carlos soy carlos (carlos está conectado)	SEND FAIL cop 2	SEND FAIL cop 2

Pruebas de la parte 2:
Como hemos explicado anteriormente, en la segunda parte de esta práctica implementamos la conexión a un servidor web a través del cual cada mensaje enviado es formateado para que solamente haya un espacio entre cada dos palabras. A continuación, presentamos algunas pruebas de funcionamiento:
Suponemos que las pruebas se ejecutan desde un usuario conectado *carlos* y el resultado es la recepción del mensaje por parte del usuario *pepe*:

Comando	**Resultado esperado**	**Resultado obtenido**
SEND pepe hola que tal pepe	MESSAGE <id> FROM jorge: hola que tal pepe END	MESSAGE 1 FROM jorge: hola que tal pepe END
SEND pepe buenas pepe	MESSAGE <id> FROM jorge:	MESSAGE 1 FROM jorge: buenas pepe

	buenas pepe END	END

Conclusiones generales de la práctica:

Esta práctica nos ha parecido muy interesante ya que hemos conseguido crear una aplicación simple de mensajería instantánea como pueden ser otras como Whatsapp utilizando dos lenguajes de programación totalmente diferentes.

Hemos tenido algún problema con las listas enlazadas ya que nos fue difícil implementar una lista enlazada dentro de otra lista enlazada y que todo funcionara como esperábamos. Para ellos tuvimos que implementar nuevas funciones que se adaptaran a los requisitos de la práctica.

También tuvimos problemas al crear el socket del servidor que se conectaba al hilo del cliente para enviarle mensajes ya que no sabíamos bien cómo implementarlo especificando la ip y el puerto. Otro problema fue que no nos funcionaba el shutdown que había que hacer en el cliente cuando se ejecutaba el unregister ya que el programa se quedaba pillado.

En general nos ha parecido una práctica entretenida con un poco de dificultad pero sin llegar a ser difícil. Creemos que hemos aprendido bastante de programación y de cómo se estructuran aplicaciones de este tipo con varios clientes y un único servidor.