Grado en Ingeniería Informática

Sistemas Distribuidos

Curso 2022/23

Universidad
Carlos III de Madrid

Autores:

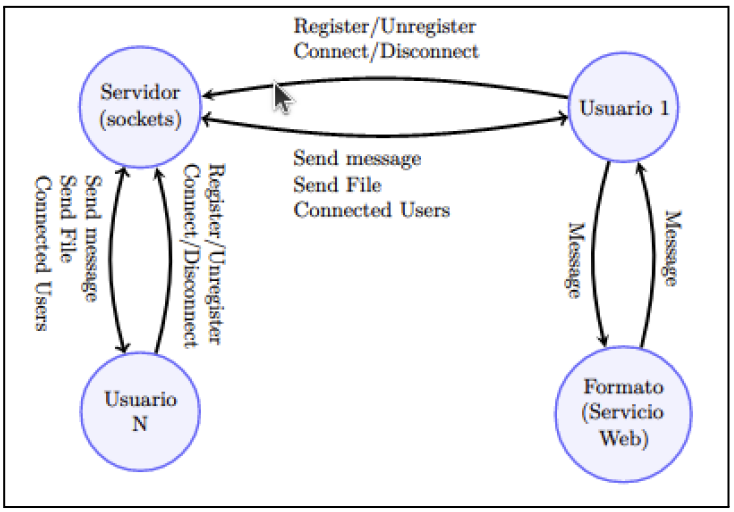
María Tapia Costa - 100451211 - g80*

Eduardo González Fernandéz - 100451296 - g80*

Índice

1. Introducción

Este documento aguardo información adicional y consideraciones acerca del proyecto final de la asignatura de Sistemas Distribuidos. Para este proyecto, el objetivo era llegar a conocer y practicar con los principales conceptos relacionados con el diseño e implementación de una aplicación distribuida utilizando distintas tecnologías y lenguajes de programación. El funcionamiento de la aplicación se muestra en la siguiente figura.



Como se puede ver en la figura de arriba, el sistema cuenta con numerosas funciones, de las cuales, han sido obligatorias implementar 6 de ellas:

- REGISTER
- UNREGISTER
- CONNECT
- DISCONNECT
- SEND
- CONNECTEDUSERS

Por lo general, la implementación de estas funciones ha sido asequible llevarlas a cabo contando con la ayuda del enunciado del proyecto junto con materiales enseñados en clase. Aquellas con las que han surgido más problemas de lo normal se explican más adelante.

2. Consideraciones del código

En este aparato del documento se destacan consideraciones acerca del proyecto. Con esto hacemos referencia a todas las funciones del sistema pero en especial a aquellas funciones que han supuesto más problemas de implementación de lo normal. También, se explica cómo hemos hecho para solventar estos problemas.

2.1 Funciones REGISTER y UNREGISTER

Las funciones REGISTER y UNREGISTER tienen el objetivo de registrar y suprimir al usuario. Para esto, cuando el usuario quiera registrarse, debe pulsar el botón REGISTER en la interfaz, aparecerá un bloque en el que debe introducir valores, distintos del valor por defecto (Text), en los tres campos disponibles. Cabe destacar que la fecha de nacimiento no puede ser introducida manualmente y debe seleccionarse mediante el botón 'Select Date'. Enviará la petición de registrarse al pulsar botón 'SUBMIT'.

Por el contrario, cuando el usuario quiera eliminar o cerrar su cuenta, debe pulsar el botón UNREGISTER en la interfaz. Para esto, no debe introducir nada más como en el caso anterior. Si todo sale bien, debe recibir un mensaje de confirmación por la pantalla de la interfaz.

2.2 Funciones CONNECT y DISCONNECT

Las funciones CONNECT y DISCONNECT tienen el objetivo de conectar y desconectar al usuario. Conectarse al servidor es imprescindible si se quiere entablar una conversación con otros usuarios, tanto como para mandarlos como para recibir aquellos que le han enviado.

Cuando un usuario quiera conectarse al servidor, es importante destacar que debe estar registrado previamente, sino esta operación no funcionará. Una vez registrado, debe pulsar el botón de CONNECT y esperar a recibir un mensaje por la pantalla de la interfaz confirmando que se ha conectado correctamente.

Cuando un usuario quiera desconectarse del sistema, debe estar previamente conectado al mismo. Una vez que está conectado, puede pulsar el botón de DISCONNECT y debe esperar a recibir el mensaje de confirmación en la interfaz.

2.3 Función SEND

La función SEND tiene el objetivo de que el usuario conectado al servidor pueda comunicarse con otros usuarios conectados o no. Cabe destacar que, aunque un usuario pueda enviar mensajes a otro no conectado, un usuario no puede mandar mensajes a un usuario que no está registrado.

Cuando un usuario quiera enviar un mensaje a otro, debe rellenar obligatoriamente los campos, 'Dest' que aparece a la izquierda de la interfaz, y el campo 'Message' un poco más a la derecha, con contenido distinto al que aparece por defecto. En caso de que cualquier campo esté vacío no se realizará el SEND. Una vez rellene todos los campos y se compruebe que el usuario destinatario está registrado, se procederá a contactar con el servicio web conversor de textos. Este servicio web se encarga de corregir los espacios introducidos por el usuario, reduciendo estos a uno. Una vez se haya formateado el mensaje, se le devuelve al cliente en cuestión y es este nuevo mensaje el que se envía al servidor. Llegados al punto de que el servidor tiene el mensaje formateado, pueden ocurrir dos casos:

- Usuario destinatario **conectado**:

  El mensaje se envía directamente a la interfaz del usuario.

- Usuario destinatario **no conectado**:

  El mensaje no se envía al destinatario sino que, se guarda en un fichero auxiliar a nombre del destinatario. De este modo, cuando éste se conecte al servidor, le llegaran los mensajes almacenados en dicho fichero.

2.4 Función CONNECTEDUSERS

La función CONNECTEDUSERS se encarga de mostrarle al usuario quien lo pide, una lista de todos los usuarios que están conectados en ese mismo instante. Cabe destacar que, esta función no se puede ejecutar si el usuario quien lo pide, no está conectado y saltará un mensaje de error en dicho caso.

Una vez el usuario esté conectado y quiera saber qué usuarios están conectados y con los que puede chatear, pulsará el botón de CONNECTEDUSERS, situado en la esquina superior derecha de la interfaz. Entonces, debe esperar a que le llegue un mensaje a la derecha de la interfaz especificando el número de usuarios conectados y los nombres por los que se puede referir a ellos en el campo 'Dest' para mandarles un mensaje.

Finalmente, queríamos añadir que esta función ha cambiado un poco respecto a las pautas consideradas en el enunciado del proyecto. Esto se debe a una pequeña incoherencia entre el desarrollo del cliente, del servidor y el protocolo de comunicación. Esto es porque, en el cliente, se especificaba que solo debías mandar un mensaje con el código de operación CONNECTEDUSERS al igual que en el protocolo de comunicación se veía como el servidor solo recibía esta cadena. Sin embargo, más adelante en el protocolo y en el desarrollo del servidor, se especifica que éste debía comprobar si el usuario quien solicita la función, está conectado. Por tanto, para satisfacer este requisito, hemos incluido también el envío del nombre del usuario para que el servidor pueda hacer las comprobaciones necesarias.

2.5 Almacenamiento de usuarios

En este apartado queremos especificar la manera en la que hemos construido el almacenamiento de los usuarios. En este caso, hemos hecho uso de archivos.txt ya que nos parecía la mejor manera de poder implementar el servicio de cara al usuario. Esto es porque, al almacenarlo en archivos, se puede desconectar el servidor y los mensajes pendientes de los usuarios, les llegarán cuando se conecten automáticamente.

Además, se ha incluido un fichero llamado 'users.txt' en el que se guarda toda la información con la que se registra un usuario. Sin embargo, los usuarios desaparecen cuando el usuario pulsa el botón mencionado anteriormente de UNREGISTER.

Por otro lado, también se ha incluido una carpeta llamada 'pending' donde se crean los archivos que incluyen los mensajes que deben recibir los usuarios no conectados. Este tipo de fichero se nombran como el usuario de manera que es imposible mandar mensajes pendientes de uno a otro incorrecto.

Finalmente, este almacenamiento de usuarios en el código lo hemos implementado con un array dinámico, declarado globalmente, que se actualiza a la vez que el archivo mencionado anteriormente ‘users.txt’. Para ser específicos, cuando un usuario pulsa UNREGISTER, se borra primero de este array dinámico y por tanto, en el archivo final. EL objetivo de este array dinámico ha sido mejorar el rendimiento del programa de tal manera que no hubiese que estar abriendo y cerrando archivos de texto contínuamente. Además, el uso de un array dinámico no supone un límite de usuarios que puede almacenar.

3. Forma de compilación

En este apartado queremos hacer referencia y explicar en detenimiento la forma de compilación del proyecto ya que pueden darse diversos errores. Para empezar, debemos abrir una terminal en nuestro ordenador y navegar hasta la ubicación del proyecto.
Podemos hacer esto usando el comando ‘cd’

3.1 Servidor

En este caso, el servidor es sencillo ya que solamente necesita la ejecución de dos comandos por separado :

```C/C++
make
./server -p <puerto_server>
```

3.2 Servicio web

En este caso, es algo más complicado ya que necesita la instalación de alguna librerías. Para instalar dichas librerías solo es necesario introducir los siguientes comandos en la terminal.

```Python
pip install Flask
```

Seguidamente, ya sería posible implementar el servicio web conversor de textos introduciendo en la terminal.

```Python
python web-service.py
```

¡IMPORTANTE!

Este servicio debe ejecutarse obligatoriamente junto con el servidor ya explicado y el cliente, a continuación. Sin este componente, la aplicación está incompleta y no funcionará como es debido.

3.3 Clientes

Finalmente, el cliente precisa de tanto el número de puerto como la IP del servidor. Por tanto, debemos ejecutar el cliente introduciendo lo siguiente la terminal.

```
Python
python client.py -s localhost -p <puerto>
```

¡IMPORTANTE!

El argumento <puerto> debe ser el mismo que ha sido introducido al ejecutar el servidor. Sino, la aplicación no funcionará como es debido.

Para verificar el correcto funcionamiento de la aplicación será necesario abrir una nueva terminal e introducir el comando para ejecutar el cliente de nuevo. De esta manera, ya tendríamos un cliente emisor y otro recepetor y viceversa.

4. Pruebas

En este apartado vamos a especificar las distintas pruebas que se han realizado a lo largo del desarrollo del proyecto, destacando para cada una de las funciones de la aplicación, casos de éxito, y de los distintos errores que se pueden dar.

1. **REGISTER**:
   - Caso de éxito: Se registra el usuario normalmente y no está registrado, devuelve un mensaje de OK
   - Caso de error, ya registrado: Se registrar con un nombre que se encuentra en users.txt. Debe obtener un mensaje de usuario en uso.
   - Caso error, conexion: Si no se reciben las respuestas de manera correcta, el sistema no puede interpretar el resultado y se muestra por pantalla.

2. **UNREGISTER**:
   - Caso de éxito: Las pruebas son similares a REGISTER. Hay un caso de éxito, en el que el usuario se encuentra registrado y se borra del sistema.
   - Casos de error: Dos casos de error, o no está registrado o hubo algún error en la comunicación.

3. **CONNECT**: Se realizaron pruebas similares.
   - Caso de éxito: el usuario no está conectado y se conecta correctamente, crea el hilo y puede recibir mensajes pendientes.

- Caso de error: Los casos de error son iguales, si está conectado lo indica y si hubo error de comunicación lo muestra en la interfaz igual.

4. **DISCONNECT**: Las pruebas de comunicación son idénticas que CONNECT. Sin embargo también se ha testado el correcto funcionamiento de detención del hilo. En cualquier caso la variable compartida cambia de valor a ejecutar disconnect y sale del hilo.

5. **SEND**: Las pruebas son varias.
   - Envio de mensaje a usuario no conectado: En este caso si toda la comunicación es correcta, se escribe en el archivo de destinatario y espera a su conexión para ser enviado.
   - Envío de mensaje a usuario conectado: En caso de que esté conectado se manda directamente y se muestra en su interfaz.
   - Caso de error: En caso de error de comunicación o que el destinatario no esté registrado se indica.

6. **CONNECTEDUSERS**: Esta función ha sido testada ejecutando en distintos momentos y comprobando con la situación real. En todos los casos se obtiene un resultado satisfactorio que concuerda con la realidad. En los casos observados el usuario que hace uso de este servicio recibe los usuarios conectados (incluido el mismo).
   - Caso de error: Se ha probado en un usuario no conectado y devuelve error, como era de esperar.

5. Conclusiones finales

En este apartado queremos reflejar nuestras conclusiones en general, a lo largo del desarrollo del proyecto.
Para empezar, nos gustaría comentar los cambios y el por qué de los cambios respecto a lo que se especifica en el enunciado de la práctica. En ese documento, había secciones ambiguas ya que no se dejaba del todo claro cómo se debía enviar el contenido de las funciones. Al final, nos decantamos por lo especificado en el protocolo de comunicación que, de manera poco precisa, comentaba que se debía hacer uso de la función send de sockets tantas veces como campos se debían enviar. Creemos que el proyecto podría haber sido más eficiente si como alternativa se plantease la concatenación de los campos.

Por otro lado, queríamos destacar que los mensajes informativos que aparece en la interfaz, no se cumple del todo con el enunciado. Este cambio ha sido con el fin de amenizar la experiencia del usuario con la aplicación. De manera que pueda saber exactamente lo que está pasando y en caso de error, no desesperarse e intentar solucionar el problema.
Algunos ejemplos de estos cambios aparecen en:

- Register, donde se especifica el nombre del usuario, ya sea afirmando el registro o informando de que su registro no ha salido bien.

Finalmente, queríamos concluir comentando que, a pesar de haber sido un proyecto más exigente de lo esperado, creemos haber cumplido con los objetivos de la práctica ya que

hemos practicado, solucionado errores y warnings desarrollando una aplicación distribuida en diferentes lenguajes.