Sistemas Distribuidos

Grado de Ingeniería en Informática. Curso 2022-2023

Práctica Final - Servicio de envío de mensajes

Grupo reducido: **82**
Letra del equipo: **J**

Práctica realizada por el alumno:

Binxian Huang: 100451010@alumnos.uc3m.es

ÍNDICE

INTRODUCCIÓN

La estructura general de la práctica es la siguiente: *cliente.py* es el fichero de código con la implementación del cliente, *server.c* es la implementación del servidor, *impl.c* es la implementación de las funcionalidades que realiza el servidor, *server.h* es el archivo de cabecera, y *web_service.py* es la implementación del servicio web encargado de convertir los mensajes que tengan varios espacios separadores en uno solo.

Los mensajes en el servidor se guardan de la siguiente manera: para cliente que se registra se crea un directorio con el nombre de su alias de registro, y los datos de registro se guardan en un archivo *user_data.txt* dentro de su directorio correspondiente, y los mensajes para ese usuario se guardan en un archivo *user_messages.txt*. Por lo tanto, cada cliente registrado tendrá un directorio con su alias, y un archivo para sus datos y otro para sus mensajes.

CÓDIGO

En el cliente la recepción de mensajes por los sockets se realiza mediante los métodos *readNumber* y *readMessage*, que leen byte a byte del socket hasta llegar al símbolo de fin '/0', se decodifican los bytes leídos y *readNumber* devuelve el número en decimal mientras que *readMessage* devuelve directamente el valor decodificado.El envío de mensajes se realiza mediante el método *sendall* de la clase socket que realiza reiteradamente envios hasta que el mensaje entero pasado por argumento se haya enviado.

En el servidor, la recepción de mensajes se realiza mediante la función readLine que lee también del socket byte a byte hasta llegar al carácter de fin '/0' y lo almacena en la variable proporcionada como parámetro. Para el envío de mensajes es similar: el mensaje a enviar se almacena como cadena de caracteres en una variable y mediante la función sendMessage se envía el mensaje.

El código de la implementación del servidor es copiado de la práctica 2.

En el archivo de implementación de las funcionalidades del servidor hay también implementadas funciones auxiliares: la funcion *client_existing* recibe el alias del cliente a comprobar, e intenta abrir el directorio con ese alias, si no lo puede abrir devuelve 0 y 1 en caso contrario; y la función *client_connected* recibe el alias a comprobar si está conectado, accede al fichero de datos del cliente y comprueba su estado de conexión, devolviendo 1 si está conectado, 0 si no y -1 en caso de algún error.

Register

La función *register*, en el cliente, crea y conecta un socket con el servidor, y manda una cadena de operación REGISTER. A continuación manda los datos de registro: el usuario, su alias y su fecha de nacimiento. Posteriormente espera a un código de respuesta del servidor, y al recibir el código cierra el socket. Si el código es 0 significa que el registro se ha realizado correctamente, si el código es 1 significa que ya hay un usuario registrado con ese alias, y error en otro caso.

En el servidor, al recibir la petición de conexión crea un hilo que la atiende, y dependiendo del mensaje de operación, espera recibir unos mensajes u otros. En este caso para el registro recibe el usuario, el alias y la fecha de nacimiento, y los guarda en una struct *client_data* que tiene como campos username, alias, birthday, identifier, online, ip y port. En este caso identifier es el identificador del último mensaje enviado, que en el registro es 0, al igual que online, ip y port. El servidor llama a la función *register_user* y le pasa como argumento la struct de los datos a guardar, la función comprueba que no exista un cliente con el mismo alias, y crea un directorio con el alias del cliente. Posteriormente guarda los valores de registro en el archivo de datos del cliente, y retorna el código 0. En caso de que ya haya un cliente registrado con dicho alias retorná el código 1 y en caso de error el código 2. Este código será el enviado y el que recibirá el cliente. Para el nombre de directorio y de archivo se reserva memoria con malloc para obtener la direccion del fichero de datos se concatena el nombre del directorio con *"/user_data.txt"* y al terminar se libera la memoria reservada.

Unregister

En el cliente la función register, al igual que la anterior, crea y conecta un socket con el servidor y envía la cadena de operación UNREGISTER. Posteriormente manda el alias del usuario que quiere realizar la baja y espera al código de respuesta del servidor. Si el código es 0 es que se ha dado de baja correctamente, 1 si el usuario con el alias no está registrado y 2 en otro caso.

En el servidor, tras el mensaje de operación espera el mensaje de alias, y llama a la funcion *unregister_user* pasandole el alias a dar de baja. En la función de baja se comprueba que el alias está registrado, y obtiene las direcciones de los ficheros de datos y de mensajes (lo mismo que con el fichero de datos en el apartado anterior pero en este caso es *"/user_messsages.txt"*). Comprueba si se puede acceder al fichero de datos, en caso afirmativo lo elimina, los mismo con el fichero de mensajes. Después intenta eliminar el directorio del alias, si lo consigue devuelve código de operación 0 dando de baja al usuario, si no lo consigue devuelve código 2 de error. Devuelve código 1 si el usuario con el alias no está registrado.

Connect

En el cliente la función connect, al igual que las anteriores, crea y conecta un socket con el servidor. También crea un socket para escuchar posibles mensajes de otros usuarios, y obtiene el puerto de ese socket. Posteriormente envía CONNECT, el alias del usuario a conectarse, y el puerto del socket de escucha, y espera el código de respuesta. Si el código es 0, se ha conectado correctamente, y crea un hilo independiente que ejecuta la función *listening_messages* para escuchar mensajes del servidor. Si el código de respuesta es 1, significa que el usuario no está registrado, en caso de ser 2 significa que el usuario ya está conectado, y 3 en otro caso.

La función *listening_messages* escucha en el socket y en un bucle infinito espera a recibir conexiones el servidor. Cuando recibe una conexión, en el socket creado espera una operación de mensaje o ack y dependiendo de la operación espera unos mensajes u otros. En caso de que en el bucle surja una excepción de error del socket, significa que el socket se ha cerrado por lo que ya no tiene que seguir escuchando y el hilo termina.

En el servidor, tras el mensaje de operación, y recibir el alias y el puerto, a través de la conexión del socket con el cliente obtiene la ip del cliente y con estos tres datos, llama a la función *connect_user*. En la funcion *connect_user,* se comprueba que el usuario con ese alias exista, se accede al archivo de datos, y se comprueba el estado del cliente: si está conectado devuelve código 2, si no está conectado cambia su valor a conectado y rellena los apartados de ip y puerto con los datos del cliente. Para la reescritura de datos se tiene un archivo temporal donde se van traspasando los datos que se leen y se van cambiando los datos que se pretenden cambiar o se dejan igual si no se pretenden cambiar. Devuelve código 0 si todo termina correctamente, 1 si el usuario existe y 3 en caso de error. Si el usuario se conecta correctamente, el servidor llama a la función *send_message*.

La función *send_message* primero comprueba que el usuario esté conectado, en caso contrario devuelve -1. Si está conectado accede al fichero de mensajes del usuario, si no puede acceder significa que no tiene mensajes por recibir y se devuelve 0. Si puede acceder, obtiene la ip y el puerto del usuario mediante la función *get_ip_port*, recorre los mensajes uno a uno y por cada mensaje que lee: establece una conexión con el cliente mediante la función *enable_connection* con la ip y puerto obtenida, le envía el mensaje y se cierra el socket de la conexión. Después comprueba si el emisor del mensaje está conectado, y si está conectado le manda mensaje de ack con la función *send_ack_to_sender.* Esto se realiza para cada mensaje que se lee, y si envía todos los mensajes correctamente, borra el archivo de mensajes y devuelve 1.

La función *get_ip_port* obtiene la ip y el puerto del usuario proporciona como parámetro, accediendo a su fichero de datos y guardando los valores en variables también pasadas por referencia a la función. devuelve 1 si se realiza correctamente y -1 en caso de error.

La función *enable_conection* recibe una ip y un puerto y establece una conexión mediante socket con el hilo de escucha del cliente, devolviendo el descriptor de socket si se ha conectado correctamente y -1 en caso contrario.

La función *send_ack_to_sender* obtiene la ip y el puerto del emisor del mensaje, con alias proporcionado como parámetro, establece una conexión con el hilo de escucha del emisor del mensaje, le envía el mensaje ack con el identificador del mensaje que recibe el receptor, y se cierra el socket de la conexión. Devuelve 1 si todo ocurre correctamente y -1 en caso de error.

Disconnect

En el cliente la función disconnect, al igual que las anteriores crea y conecta un socket con el cliente, envía el mensaje de desconexión y el alias del usuario, y espera el código de respuesta. Si el código es 0 significa que se ha desconectado correctamente y cierra el socket de escucha de mensajes creado en la conexión, y por tanto el hilo de escucha. En caso de ser el código de respuesta 1, significa que el usuario a desconectar no existe, si es 2 es que el usuario ya está desconectado y 3 en otros casos.

En el servidor, tras recibir el mensaje de operación y el alias del usuario a desconectar, llama a la función *disconnect_user*. En la función *disconnect user*, se comprueba que esté registrado, y se devuelve el código 1 en caso contrario. Después, al igual que en la conexión, se accede al fichero de datos del usuario y se comprueba el estado de conexión: si no está conectado se devuelve el código 2, si está conectado se cambia su estado a desconectado y los valores de ip y puerto a nulo, devolviendo 0, y devuelve 3 en caso de error.

Send

En el cliente, se realiza lo mismo con las funciones anteriores, se crea y conecta un socket con el servidor. En este caso también se conecta con un servidor de servicio web, en ejecución en este caso con anterioridad en la máquina local, que elimina los espacios entre

palabra y palabra excesivos y dejando solo uno, y devolviendo el mensaje transformado, que será el que se pase al servidor. Tras esto espera el código de respuesta: si es 0 es que el mensaje se ha enviado con éxito y espera el identificador del mensaje enviado que será la respuesta del servidor, si alguno de los usuarios no existe o el emisor no está conectado el código es 1, y 2 en otros casos.

En el servidor, tras recibir el mensaje de operación y el alias del emisor, el alias del receptor, y el mensaje, llama a la función *save_message* para guardar primero el mensaje. En la función *save_message* se comprueba que ambos usuarios estén registrados y el emisor está conectado, devolviendo código 1 en caso contrario. Después obtiene el identificador que le corresponde al mensaje, mediante la función *get_identifier*, y guarda el mensaje en el archivo de mensajes del receptor mediante la estructura *message_data*, con los campos sender, identifier y message. Si se realiza correctamente devuelve 0, y 2 en caso de error. Si el mensaje se guarda correctamente, llama a la función *send_message* para enviar el mensaje guardado, la misma que la explicada anteriormente.

La función *get_identifier* accede al archivo de datos del usuario con el alias recibido como parámetro, obteniendo el último identificador de mensajes, si es el máximo se vuelve a poner en 1 y si no se añade una unidad y se cambia el valor del identificador por el nuevo. Si la función se realiza correctamente devuelve el identificador y -1 en caso contrario.

ConnectedUsers

En el cliente la función connectedUsers, al igual que las anteriores crea y conecta un socket con el cliente, envía el mensaje de petición de usuarios y espera el código de respuesta. Si el código es 0 significa que se ha realizado correctamente, y espera un mensaje del número de usuarios conectados, y en función del número de usuarios conectados realiza tantas esperas de mensajes en un bucle hasta recibirlos todos. En caso de ser el código de respuesta 1, significa que el usuario de la petición no está conectado y 2 en otros casos.

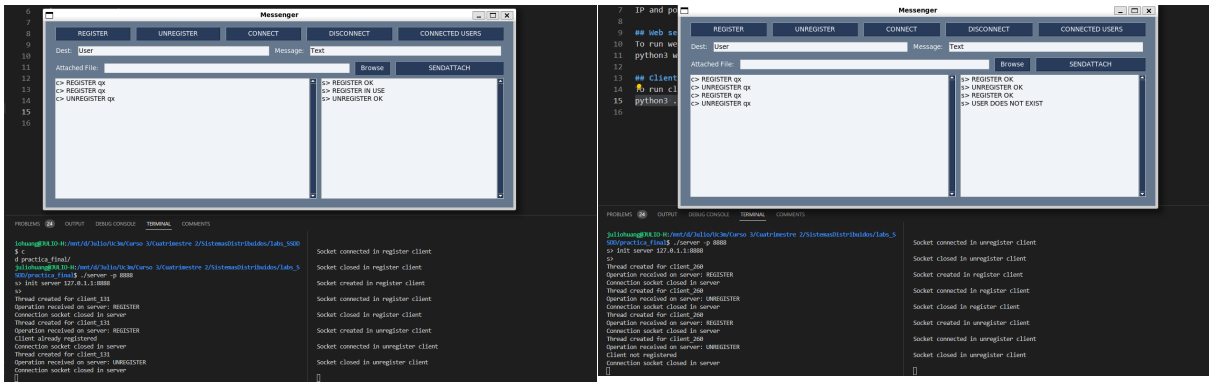
En el servidor, tras recibir el mensaje de operación y el alias del usuario, llama a la función *connected_users*. En la función *connected_users*, se comprueba que esté registrado, y se devuelve el código 2 en caso contrario. Se comprueba que esté conectado, devolviendo el código 1 en caso contrario. Si está conectado, recorre el directorio actual abriendo todos los directorios correspondientes a usuarios registrados, accediendo a sus ficheros de datos y comprobando su estado de conexión. Si están conectados, se guarda su alias en una lista y se incrementa el número de usuarios conectados. Si termina correctamente devuelve el código 0 y 2 en caso contrario. Si termina correctamente, se envía el número de usuarios conectados como mensaje al cliente y se recorre la lista de los alias enviándolos uno a uno al cliente.

COMPILACIÓN

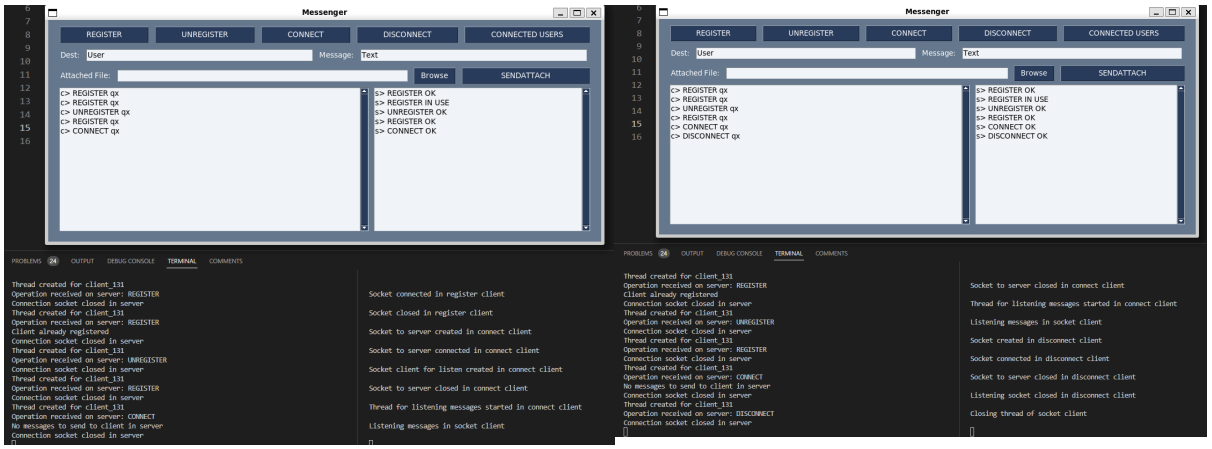
Compilación de servidor con fichero Makefile.
Ejecución del servidor: ./server -p <port>
La terminal del servidor mostrará la dirección y puerto en la que escucha
Ejecucion del archivo python para el servicio web: python3 web_service.py
Ejecutar cliente con la ip y puerto proporcionada por el servidor:
python3 ./client.py -s <ip> -p <port>

PRUEBAS

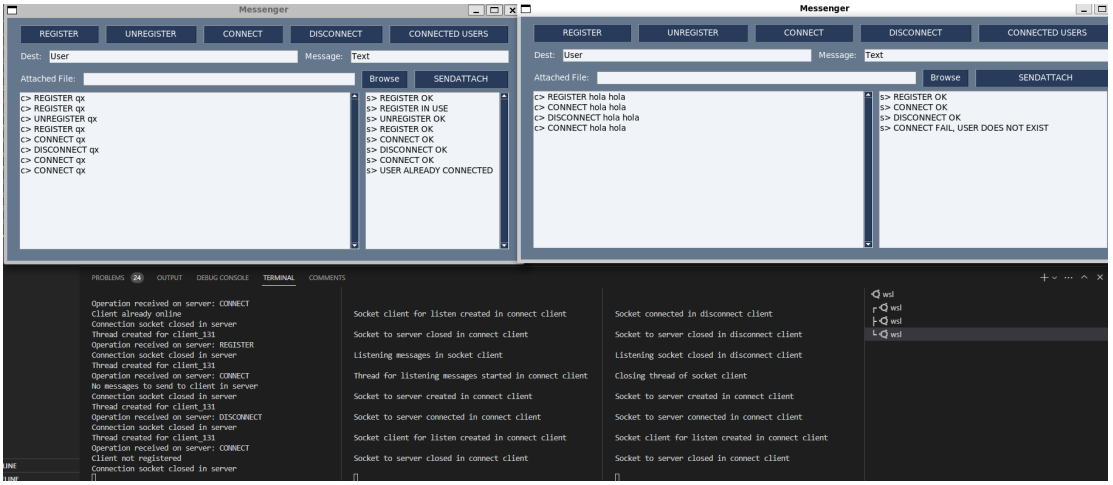
Registro y baja de usuario, y baja de usuario al eliminar el directorio de registro::



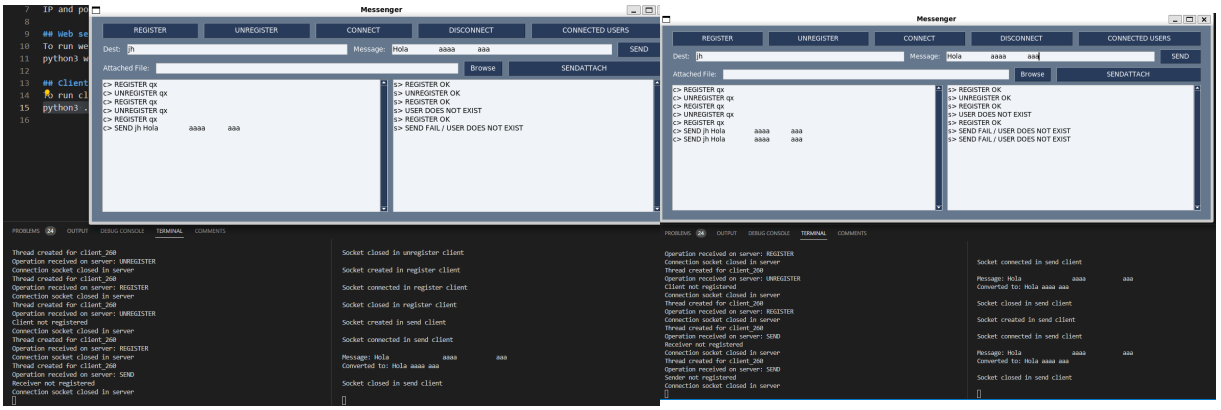
Conexión y desconexión de usuario:



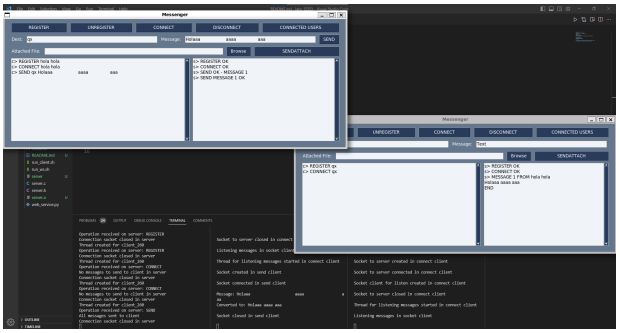
Caso en el que se registra otro usuario “hola hola”, se elimina el directorio de registro y se intenta conectar:



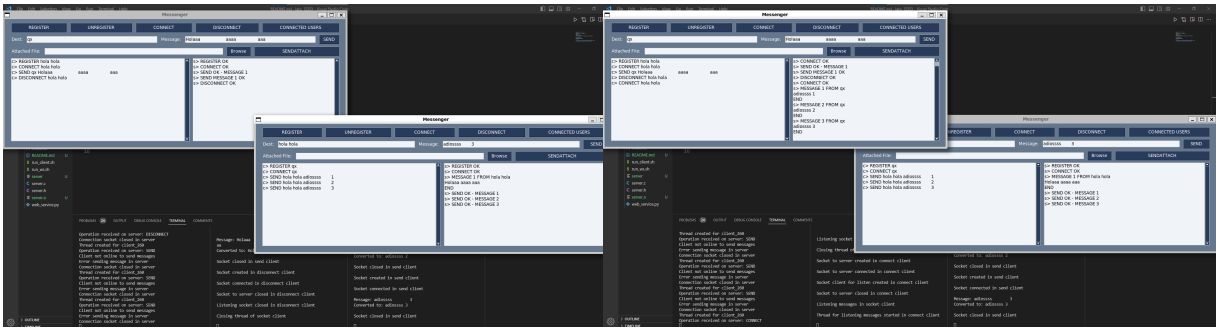
Enviar mensaje a usuario no registrado y si se elimina el directorio del usuario emisor:



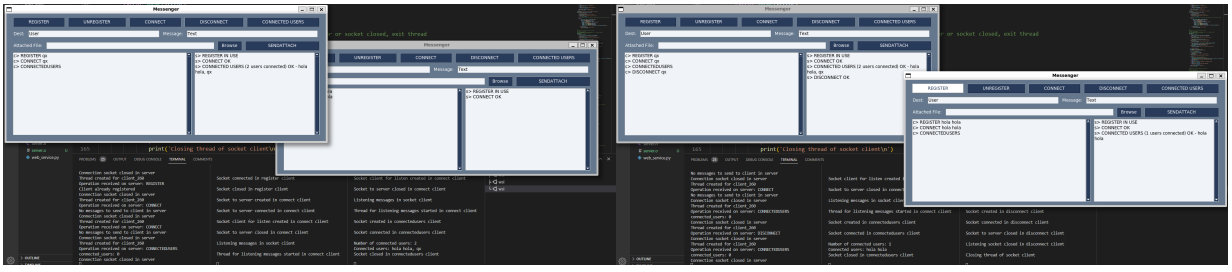
Mensaje a receptor conectado y emisor conectado:



Mensaje a receptor desconectado, y posterior conexión:



Connected users:



CONCLUSIONES

En conclusión la práctica me ha parecido muy interesante, sobre todo porque es un conjunto de todo lo estudiado en el cuatrimestre, y cuando la implementación funciona correctamente sabes que has aprendido bastante de lo enseñado.

En cuanto a las dificultades, no ha sido tanto en esta práctica final ya que, como es parecida a las interiores, las dudas más cruciales ya se resolvieron en su momento por lo que al realizar la misma parte ya no surgen las misma dudas. En concreto la parte que más a costado ha sido como diseñar el código para todo el paso de mensajes, si hay que enviar los mensajes si está conectado, o si hay que guardarlos para enviarlos cuando se conecten.

En general, no ha sido muy complicada teniendo las bases anteriores, aunque sí un poco bastante larga, y juntado con las demás prácticas de otras asignaturas también hay bastante presión.