Sistemas distribuidos:

Práctica Final

Diseño e implementación de un sistema peer-to-peer

Belén Gómez Arnaldo

100472037

Ana Maria Ortega Mateo

100472023

12 de mayo de 2024

ÍNDICE

Arquitectura

La arquitectura del trabajo, se divide en cuatro componentes fundamentales: servidor principal, cliente de python, servidor web y servidor RPC cuya interfaz es server_hora.x.

- Servidor principal: La implementación del servidor principal se encuentra en "servidor.c" (lenguaje C). El servidor se encarga de manejar las distintas peticiones de los clientes. Recibe una petición con la operación que ha de realizar y los distintos parámetros que necesita para realizar dicho procedimiento. El servidor puede almacenar, modificar o entregar los datos al cliente según sea necesario. Además, actúa como intermediario facilitando la comunicación entre los clientes proporcionando la dirección del cliente remoto para permitir la transferencia de archivos. La comunicación con los clientes se realiza mediante sockets. El socket del servidor se mantiene siempre escuchando peticiones en el puerto que se especifica como argumento en la terminal. Para las funciones send y recv utilizamos las funciones proporcionadas por "<sys/socket.h>", para poder establecer la comunicación y mandar y recibir los mensajes a través de sockets.

- Cliente: La implementación del cliente se encuentra en "client.py" (lenguaje python). El cliente es el que interactúa con el servidor y realiza peticiones para poder realizar las diferentes funcionalidades del sistema. Cada cliente es independiente y pueden conectarse al servidor para realizar funcionalidades específicas. También pueden comunicarse mediante sockets con otros clientes para enviar archivos entre ellos. El servidor simplemente aportará la información necesaria para que se pueda realizar dicha comunicación. Este proceso implica que el cliente actúe como un nodo en la red, aprovechando los principios de la comunicación punto a punto (peer-to-peer) para establecer una conexión directa con otro cliente permitiendo poder intercambiar los archivos entre sí en una conexión cliente-cliente independiente al servidor.

- Server_hora.x: Archivo de definición de la interfaz RPC que se utilizará para realizar llamadas a procedimientos remotos en el programa llamado SERVER_HORA. Contiene, un número de programa, un número de versión del programa y el nombre y número de procedimiento. Se define en él, las variables de entrada y de salida de nuestra función. A partir de esta interfaz se generan todos los archivos necesarios para la comunicación RPC. El cliente generado no será utilizado, en su lugar, utilizaremos el archivo server.c, en el que se llamará al servidor RPC cada vez que se reciba una petición. Este servidor imprimirá los valores pasados como parámetros en una estructura.

- Servidor Web: Este servidor se encuentra en el archivo servidorWeb.py (lenguaje python). Este servidor tiene una única función que devuelve una cadena de caracteres con la fecha y hora actual. Cada vez que un cliente quiera hacer una petición al servidor principal primero obtendrá la hora y fecha actual de este servidor web y se la mandará al servidor principal.

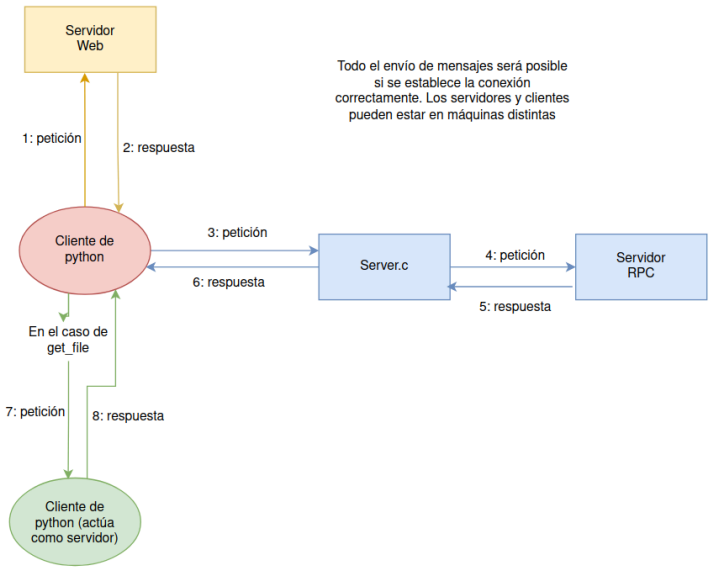

Diagrama de la estructura

Estructura de los mensajes

La comunicación entre el cliente y el servidor principal se basa en un mecanismo de petición y respuesta. Cuando un cliente necesita realizar una acción o solicitar información al servidor, envía una petición al servidor. El servidor procesa esta petición y envía una respuesta de vuelta al cliente.

- Peticiones del cliente al servidor principal: Para realizar una petición la estructura de los mensajes dependen de la funcionalidad que el cliente quiere realizar. Para realizar cualquier tipo de acción es necesario mandar al servidor la operación a realizar (cadena de caracteres). Las funciones REGISTER, UNREGISTER, DISCONNECT y CONNECT solo necesitan un nombre de usuario, que se introduce como argumento y se manda como una cadena después de la operación. En el caso de LIST_CONTENT se envía el nombre de usuario conectado y el nombre que se introduce como argumento. En el caso de LIST_USERS, se envía la operación y el nombre de usuario conectado (str). En funciones con más parámetros como PUBLISH o DELETE, se mandan los parámetros separados por espacios para posteriormente poder separarlos y coger cada uno de los argumentos. Todas las peticiones son cadenas de caracteres, sólo difieren en el número de parámetros a mandar. Para que el servidor reconozca el final de una petición es necesario mandar como terminación un “/0” que le indicará cuando tiene que terminar de esperar a recibir peticiones.

Además, con la segunda parte de la práctica también se envía en la petición la fecha y hora que se obtiene del servidor web. Esto se envía en la cadena justo después de la operación. Se separan todos los parámetros por espacios para poder extraerlos fácilmente en el servidor.

Para manejar las distintas peticiones de los clientes el servidor almacenará, modificará o eliminará el contenido de los ficheros, que es la estructura elegida para registrar las acciones de cada procedimiento. Se ha optado por esta estructura de almacenamiento porque los datos permanecen intactos incluso después de que el servidor finalice. Esta característica es esencial para garantizar la integridad y la disponibilidad de los datos a largo plazo. Otra ventaja es la facilidad para acceder y modificar archivos y la gran cantidad de información que estos permiten almacenar.

- **Respuestas del servidor principal:** Las respuestas que le manda el servidor al cliente son, en la mayoría de los casos, cuatro bytes (un entero), indicando el estado de la funcionalidad realizada. Para ello, el servidor serializa el entero (big endian), para que pueda ser reconocido entre diferentes sistemas. En algunos casos, como por ejemplo LIST_CONTENT, se tienen que mandar las publicaciones de un usuario. Por ello se manda primero la cantidad de publicaciones realizadas (entero serializado de cuatro bytes), para que el cliente espere a recibir la cantidad exacta de publicaciones. Posteriormente se mandan las cadenas de caracteres de la información a enviar separando cada parámetro con el delimitador "/0", para que el cliente conozca el final de cada argumento recibido.

- **Comunicación cliente-cliente**: la estructura de los mensajes es similar a la del servidor y cliente. Se le envía al cliente el nombre del archivo que se quiere descargar (str) y el otro cliente se encargará de mandar primero un entero serializado del estado de su petición seguido de otro entero serializado (big-endian) con el tamaño del archivo. Después se irá mandando el contenido del archivo, de 1024 a 1024 bytes hasta que se recibe todo el archivo.

- **Comunicación con el servidor RPC:** cada vez que al servidor le llega una petición, actúa como cliente mandando un mensaje al servidor RPC. El servidor RPC solo tiene una función que recibe como argumento una estructura (definida en el archivo server_hora.x) con los datos que tiene que imprimir. En el enunciado se pide que se manda una petición al servidor RPC cada vez que se recibe una petición en el servidor principal, por lo que se manda la petición aunque el servidor principal luego devuelva algún error (como usuario no registrado o no conectado). En las funciones de publish y delete, junto con la operación se manda el archivo que se publica o se elimina para que se imprima también. El servidor devuelve un número entero indicando que se ha realizado la petición correctamente.

Protocolos de la aplicación

Nuestra aplicación se basa en una comunicación por sockets utilizando el protocolo TCP, entre clientes en Python y un servidor en C. En primer lugar, el protocolo TCP proporciona un alto nivel de fiabilidad en la transmisión de datos. Esto significa que los datos enviados por un cliente llegarán al servidor de manera íntegra y en el mismo orden en que fueron enviados. Esto es necesario para nuestra

aplicación, asegurando que no se pierde ningún tipo de información al realizar el envío de un mensaje. El protocolo TCP también es muy bueno para evitar la congestión del servidor y la pérdida de los mensajes, teniendo un mecanismo de control de flujo equilibrado y eficiente, algo necesario para nuestra aplicación para evitar sobrecargar al servidor cuando varios clientes están conectados realizando acciones. Además, TCP es altamente compatible con una amplia variedad de sistemas operativos. Esta compatibilidad garantiza que nuestra aplicación funcione de manera consistente y confiable en diferentes entornos.

Los pasos para compilar y ejecutar el código se encuentran en el archivo README.

Operaciones de cada componente de la arquitectura

Tanto el servidor como el cliente realizan distintas operaciones para poder llevar a cabo cada funcionalidad. A continuación explicaremos la implementación de estas funcionalidades en cada uno de los componentes de nuestro sistema.

Cliente

Para realizar cada operación se nos ha proporcionado una shell que recoge los argumentos pasados por terminal y llama a la función correspondiente. Antes de nada, se obtiene la dirección ip y puerto pasados por terminal, que permitirán poder conectar el cliente con el servidor, siempre y cuando el puerto sea el mismo. Como pasos generales de todas las funciones, se crea el socket con el protocolo TCP , utilizando el puerto y la ip pasados por terminal. Se conecta el socket al servidor. Se envía al servidor toda la información necesaria para realizar la función, separando por espacios y como parte final del mensaje un “/0”. El cliente posteriormente, esperará a recibir el estado de su petición teniendo en cuenta los distintos escenarios posibles y deserializando el número que indica el estado de su acción. Finalmente, siempre se cierra el socket. A continuación, se explica las diferencias que tienen las funciones, en caso de que todo haya ido bien:

1. Función Register

Primero se comprueba si el cliente estaba conectado para no permitir el registro de otro cliente si no se ha desconectado. Esto se hace fundamentalmente para mantener la consistencia del sistema y que el servidor no se confunda a la hora de que cliente esté realizando dicha acción. También se crea una carpeta para cada cliente que se registra, de forma que pueda publicar los contenidos de su carpeta. Además se guarda el usuario registrado para poder identificarlo en otras funciones.

2. Función Unregister

Borra toda la carpeta del usuario correspondiente y se pone el valor de la variable de usuario registrado a “None”.

3. Función Connect

La función connect permite a los clientes conectarse al servicio. Se crea un socket, asignándole un puerto válido y enlazando con una dirección disponible. Este será el encargado de escuchar las peticiones de otros clientes. Se almacena el cliente conectado para utilizarlo en otras funciones y, si el cliente registrado era distinto, se iguala al cliente conectado. Hemos decidido hacer esto porque se puede conectar un usuario que sea distinto al último que se ha registrado, pero usamos el cliente registrado para identificar al usuario que está realizando las operaciones. Si se ha hecho un connect el usuario que está realizando las operaciones es el que se acaba de conectar.

Cuando se llama a esta función también se crea un hilo que será el encargado de escuchar y atender las peticiones de descarga de ficheros de otros usuarios, ejecutando la función atender_peticiones. Hemos decidido que todos los hilos sean detached (daemon=True), no dependen el uno del otro por lo que no tendrán que realizar un join para esperar a que los demás hilos se ejecuten.

4. Función Atender_peticiones

Esta función se encarga de aceptar conexiones entrantes por el socket. Cada vez que llega una petición se crea un hilo para evitar bloquear el hilo master del programa. Se realiza mutex en la variable conectado, para proteger el acceso a esta variable cuando dos clientes intentan conectarse a la vez. Las peticiones de los clientes se llevan a cabo en atender_cliente.

5. Función Atender_cliente

Esta función recibe como parámetro el socket del cliente para poder mandar y recibir información. También recibe el nombre de fichero que el usuario está interesado en descargar. Verifica si el archivo existe y en tal caso, manda el archivo al cliente, incluyendo su tamaño (para que sepa la longitud del mensaje) y contenido. Se utiliza un mutex para garantizar que el acceso al archivo y al socket sea seguro y evitar condiciones de carrera al entrar en recursos compartidos. En caso de éxito devuelve 0 y en caso de error 1.

6. Función Disconnect

El cliente dejará de estar conectado. Antes de nada se comprueba que haya algún usuario conectado, si no se mostrará un mensaje indicando que no hay usuario que desconectar. Además se comprueba que el usuario que se manda como argumento para desconectar sea el usuario actual. Esto lo hacemos para

evitar que un usuario pueda desconectar a otro usuario que esté haciendo otras operaciones en el servidor, cada usuario solo se puede desconectar a sí mismo.

7. Función Publish

Antes de nada se comprueba si hay algún usuario registrado. Esto lo hacemos porque en esta y otras funciones es necesario que se haya identificado un usuario. En estas funciones no hay que especificar un usuario como parámetro, pero el servidor necesita recibir el nombre de usuario de quien realiza la operación. Si nada más ejecutar el cliente se hace esta función, no se habrá registrado ni conectado ningún usuario por lo que el sistema no sabrá qué usuario está realizando la operación. En ese caso se muestra un mensaje. Este caso es distinto al de que no se haya conectado el usuario. El usuario se puede haber registrado pero no conectado, en cuyo caso la aplicación devolverá un resultado distinto.

Una vez comprobado esto se comprueba que el fichero que se quiere publicar se encuentra en el directorio del usuario. En caso contrario el fichero no existe y no se puede publicar.

8. Función Delete

Se eliminará el fichero en el directorio correspondiente. Se hace la misma comprobación sobre el usuario que está realizando la operación que en la función publish.

9. Función List Users

Se hace la misma comprobación sobre el usuario que está realizando la operación que en la función publish. Esta función recibe primero la cantidad de usuarios conectados, para luego esperar a recibir la información de la ip y del puerto de cada usuario. Se imprimen los mensajes recibidos por terminal, según se indica.

10. Función List Content

Se hace la misma comprobación sobre el usuario que está realizando la operación que en la función publish. Esta función recibe la cantidad de publicaciones del usuario especificado, para luego esperar a recibir la información de cada publicación. Se imprime la información recibida por terminal, según se indica.

11. Función Get File

Esta función manda una petición al servidor solo para conocer la dirección ip y puerto del usuario remoto al que desea conectarse. Si se reciben correctamente estos datos, se crea el socket y se conecta con el socket del usuario remoto para poder mandarle peticiones. El cliente envía el nombre del fichero de su interés. Si todo ha ido bien, el cliente procede a recibir el tamaño del archivo. Posteriormente, se

lee el contenido del archivo en bloques de 1024 bytes hasta que se completa la recepción del archivo. El archivo se guarda en el directorio del usuario registrado en el servidor.

Servidor principal

En server.c se realiza la implementación de funciones para gestionar y atender las peticiones. En el servidor primero se comprueba si los argumentos pasados por la terminal son correctos. Posteriormente se asignan las variables para el socket y se crea el socket TCP. Configuramos el socket y le asignamos el puerto, cogiendo el tercer argumento pasado por terminal. Este argumento se pasa a entero con strtol para poder hacer manejo de errores. Con bind, asociamos la dirección al socket y se hace un listen() para preparar el socket a aceptar conexiones.

A continuación se obtienen las variables de entorno del programa. Esto se hace para después poder configurar la conexión con el servidor RPC. Como no se especifica en el enunciado, hemos decidido que para ejecutar el servidor haya que especificar las variables de entorno como en los ejercicios anteriores. De esta forma, el servidor se ejecutará con un comando como este: env IP_TUPLAS=IP, donde la dirección IP será la dirección donde se encuentre el servidor RPC.

Después hacemos un bucle while, para que cada vez que llegue una petición por parte de un cliente y se acepta la conexión, se crea un socket de la conexión aceptada. Se crea un hilo por cada petición que llama a la función execute command, pasando el socket de la conexión aceptada para poder hacer la recepción y envío de mensajes necesarios para las funciones. Los hilos son independientes (detached), no dependen el uno del otro por lo que no tendrán que realizar un join para esperar a que los demás hilos se ejecuten.

1. Función execute_command

En esta función se llama a la función correspondiente que el cliente quiere realizar. Se protege la copia del socket con mutex, para evitar condiciones de carrera y el servidor esperará a recibir todo el mensaje que el cliente le envía, que son los parámetros necesarios de cada operación. En función de la operación recibida, se llamará a dicha función. Envía de vuelta el resultado que devuelve la función, exceptuando algunos casos, para mantener el orden en el que se envían las respuestas. Finalmente, cierra el socket y el hilo.

Todas las funciones llaman a la función “llamar_rpc”. Tras recibir la operación se llama a esta función, pasándole como parámetro el nombre del usuario, la operación (si es necesario junto con el nombre del fichero), la fecha y hora (obtenidos a través del servidor web). Esta función realiza los siguientes pasos:

- Se declaran variables necesarias para la comunicación RPC por parte del cliente, definiendo el estado de retorno, la entrada de los parámetros de la función o la respuesta.
- Obtenemos la dirección IP con nuestra función ObtenerIP(), que simplemente devuelve la IP anteriormente almacenada en la variable global y la asignamos a la variable host, para definir la dirección del servidor.
- Creamos el cliente RPC, pasando, la dirección del servidor, el nombre del programa y de la versión (definidas en la interfaz) y utilizamos el protocolo "tcp".
- Se asignan los datos de entrada.
- Se llama a la función remota. El resultado que devuelve la función se guarda en la variable result_1. En caso de error se almacenará -1 y en caso de éxito 0.
- Se destruye el cliente RPC.

Hay funciones que utilizan la mayoría de procedimientos, como buscar registrado y buscar conectado, que buscan en los ficheros, línea por línea, el nombre de usuario.

Cada vez que se accede a una función y se utilizan recursos compartidos en memoria, se protegen estos accesos con un mutex para evitar condiciones de carrera. Cuando se deja de acceder, el mutex se desbloquea, protegiendo solamente la situación en la que se accede a un recurso de memoria compartida.

2. Función register

Comprueba que el nombre de usuario no tenga una longitud de más de 256 caracteres. Busca si el nombre de usuario ya existe o no en usuarios.txt. En caso de que no exista se almacenará en el fichero el nombre del nuevo usuario y el cliente será registrado.

3. Función unregister

Recorre el fichero de registrados (usuarios.txt) y va guardando las líneas en las que no se encuentra ese nombre en un fichero temporal. En caso de que lo encuentre se copiará en usuarios.txt, línea por línea el fichero temporal, no añadiendo al usuario encontrado. Se borrará también si aparece en los ficheros de conectados y se eliminará de publicaciones todas sus publicaciones.

4. Función connect

Se comprueba si el usuario está registrado y conectado. Se almacenará en el archivo conectados.txt el nombre del usuario conectado, la ip y el puerto de dicho cliente.

5. Función publish

Se comprueba si el usuario está registrado y conectado. Se lee línea por línea el fichero publicaciones.txt para comprobar que no se ha publicado ningún fichero con ese mismo nombre. Si no existe ese fichero, se almacenará en el fichero el nombre del usuario, del fichero y una breve descripción.

6. Función delete (borrar)

Se comprueba si el usuario está registrado y conectado. Recorre el fichero de publicaciones y va guardando las líneas en las que no se encuentra el nombre del fichero, en uno temporal. En caso de que lo encuentre se copiará en publicaciones.txt, línea por línea el fichero temporal, no añadiendo la publicación, por lo tanto, eliminandola. Para eliminarlo, el nombre de usuario conectado tiene que ser el mismo que el de la publicación que se desea eliminar, evitando de esta manera que otro usuario pueda eliminar tus publicaciones.

7. Función list_users

Se comprueba si el usuario está registrado y conectado. Envía por el socket el resultado. Posteriormente, cuenta el número de líneas del fichero conectados.txt (corresponde con el número de usuarios) y pasará el número al cliente para saber de cuantos usuarios tiene que esperar la información. Se manda finalmente por el socket mensaje por mensaje, el usuario, la ip y el puerto.

8. Función list_content

Se comprueba si el usuario está registrado y conectado. Envía por el socket el resultado. Posteriormente, cuenta el número de líneas en las que aparece el usuario remoto y pasará el número al cliente para saber de cuantos publicaciones tiene que esperar la información. Se mandará por socket el nombre del fichero y la descripción de cada fichero publicado de ese usuario.

9. Función disconnect

Recorre el fichero de conectados y va guardando las líneas en las que no se encuentra ese nombre en un fichero temporal. En caso de que lo encuentre se copiará conectados.txt, línea por línea el fichero temporal, no añadiendo al usuario encontrado.

10. Función get_file

Se comprueba si ambos usuarios están registrados y conectados. Envía por el socket el resultado. Si todo va bien, se enviará por el socket la dirección ip y el puerto del usuario remoto. Se ha decidido que el servidor se encargue de esto porque hemos considerado que dos usuarios deberían estar conectados y registrados y que la información de la ip y el puerto debería ser aportada directamente por el servidor

para ayudar a dos usuarios conectarse y simplificar al cliente dicho proceso (no hay ninguna especificación de la obtención de ip y puerto en el enunciado).

Todas las funciones envían al cliente el estado de su acción (serializando el número), por lo que se tienen en cuenta todos los escenarios posibles.

Servidor RPC

Se ha definido la interfaz de RPC en "server_hora.x". En esta interfaz se define el nombre del programa y el de la versión, junto con la función de impresión y los parámetros de entrada (la estructura) y un entero de salida que indicará el estado de la comunicación. Se crean todos los ficheros y en nuestro servidor principal añadimos la función "llamar_rpc". Esto hace que el servidor principal ahora actúa como cliente mandando una estructura con los datos necesarios al servidor_hora_server.c. Este se encarga de imprimir por pantalla los distintos elementos pasados por la estructura y devolverá en el resultado 0 en caso de éxito.

Servidor Web

Se ha configurado un servidor de servicios web que proporciona la hora actual en respuesta a acciones realizadas. La función get_current_time devuelve la hora actual en un formato específico. Los clientes llaman a este servicio SOAP para obtener la hora actual del servidor web. Al ejecutar el servidor web se tiene que especificar la dirección donde se va a ejecutar. De esta forma los clientes deben también especificar la dirección del servidor web al que conectarse. Esto permite que se conecten clientes desde una máquina distinta a la del servidor web.

PRUEBAS

Todos los comandos que hemos utilizado se encuentran en el archivos pruebas.txt numeradas de la misma forma que a continuación. Cabe destacar que si se introduce un comando que no coincide con ninguna operación o no se pasa el número de argumentos necesario se devuelve un error en el cliente y no se envía ninguna petición al servidor.

1. **Registrar un usuario**: se registra un nombre de usuario válido. Se crea una carpeta para este usuario y se añade al fichero de usuarios.txt. El servidor imprime un mensaje de quien ha realizado la operación y de que se ha realizado con éxito. El servidor RPC imprime la información correspondiente a la petición.
2. **Registrar un usuario con un nombre en uso:** si se intenta registrar un usuario con un nombre que ya se encuentra en el fichero de usuarios.txt se devuelve un mensaje de que el nombre ya está en uso. No se vuelve a escribir en el fichero ni se crea otro directorio.

3. **Borrar un usuario**: se borra el nombre del usuario del fichero de usuarios.txt y, si estaba conectado, se elimina de conectados.txt. También se elimina su directorio con todos los archivos que tenía. Por lo tanto, si tenía alguna publicación en el archivo de publicaciones se elimina tambié.

4. **Borrar un usuario que no está registrado**: si se intenta borrar un usuario que no está registrado se devolverá un mensaje de error ya que el usuario no existe.

5. **Conectar un usuario registrado**: si se conecta un usuario que ya está registrado, el servidor imprimirá un mensaje de que se ha conectado correctamente y se escribirá en el fichero de conectados.

6. **Conectar dos usuarios a la vez**: si ya hay un usuario conectado no se permite conectar a otro usuario. Primero habrá que desconectar al primer usuario para poder conectar uno nuevo. Se mostrará un mensaje por la terminal del cliente y la petición no llegará al servidor.

7. **Conectar un usuario que no está registrado**: si se intenta conectar un usuario que no está en el archivo usuarios.txt el servidor devolverá un mensaje de error de que el usuario no existe y no se conectará.

8. **Conectar un usuario que ya está conectado**: este caso solo se puede dar si se intenta conectar otro usuario, es decir, si se ejecuta en otra terminal. Como hemos dicho en la prueba 6, no se pueden conectar dos usuarios a la vez. Esto implica que desde un mismo cliente no puede surgir este error ya que para poder hacer un connect se tiene que haber desconectado primero el usuario. Sin embargo, sí que puede surgir en otra terminal. Si un cliente hace “CONNECT CLIENTE” y se abre otra terminal y se intenta conectar el mismo usuario el servidor devolverá el error de que el usuario ya está conectado.

9. **Desconectar un usuario que no existe**: se muestra un mensaje de error de error ya que el usuario no está registrado, no se encuentra en el archivo usuarios.txt.

10. **Desconectar un usuario que no está conectado**: se muestra un mensaje de error por la terminal ya que se ha intentado desconectar un usuario que no está conectado.

11. **Desconectar un usuario distinto al que está conectado: s**i solo hay un usuario conectado (solo una terminal) esto no podría suceder porque solo se podría desconectar ese mismo usuario. Sin embargo, si hay varios usuarios conectados, un usuario no podría desconectar a otro usuario distinto. Por eso antes de hacer la operación se comprueba que el usuario que se quiere desconectar sea el mismo que está conectado. En caso contrario se devuelve un mensaje de error.

12. **Desconectar un usuario**: cuando se desconecta un usuario se elimina del archivo de conectados.txt. Cuando se ejecuta el comando “quit” se desconecta también el usuario que hubiera conectado, si lo había.

13. **Desconectar un usuario sin haberse conectado**: en este caso también se muestra un mensaje de error de que no hay ningún usuario conectado.

14. **Publicar un contenido sin estar registrado (sin estar identificado:** como hemos explicado anteriormente, si el usuario no se ha identificado (registrado o conectado) no se puede realizar esta función. Se muestra un error de que el usuario no se ha identificado.

15. **Publicar un contenido sin estar conectado:** se muestra un error de que el usuario no está conectado y no se publica ningún archivo.

16. **Publicar un contenido que no exista en la carpeta del usuario:** un usuario no puede publicar un archivo que no tenga en su directorio porque en realidad ese archivo no existe. Si otros usuarios quisieran acceder al archivo mediante la función get_file, daría errores porque el archivo estaría en publicaciones.txt pero no existiría.

17. **Publicar un contenido correctamente:** cuando se publica un archivo correctamente se añade al archivo de publicaciones.txt con el usuario que lo ha publicado y una pequeña descripción.

18. **Publicar un contenido con un título repetido:** no se puede publicar un archivo con un título que ya exista en el archivo de publicaciones por lo que se devuelve un error.

19. **Borrar un contenido sin estar registrado (sin estar identificado):** como hemos explicado anteriormente, si el usuario no se ha identificado (registrado o conectado) no se puede realizar esta función. Se muestra un error de que el usuario no se ha identificado.

20. **Borrar un contenido sin estar conectado:** se muestra un mensaje de error de que el usuario no está conectado.

21. **Borrar un contenido que no esté publicado:** se muestra un mensaje de error de que el archivo no está publicado.

22. **Borrar un contenido publicado por otra persona:** un usuario solo puede borrar archivos que haya publicado él mismo, por lo que se mostrará un mensaje de error de que ese archivo no está publicado ya que no lo ha publicado él.

23. **Borrar un contenido correctamente:** cuando se borra correctamente un archivo se elimina del archivo de publicaciones.txt y se elimina del directorio del usuario.

24. **Hacer un LIST_USERS sin estar registrado (sin estar identificado):**como hemos explicado anteriormente, si el usuario no se ha identificado (registrado o conectado) no se puede realizar esta función. Se muestra un error de que el usuario no se ha identificado.

25. **Hacer un LIST_USERS sin estar conectado:** se muestra un mensaje de error de que el usuario no está conectado.

26. **Hacer un LIST_USERS correctamente:** se mostrarán todos los usuarios conectados mostrando su dirección IP y el puerto en el que se ha creado el socket para que se puedan conectar otros clientes.

27. **Hacer un LIST_CONTENT sin estar registrado (sin estar identificado):** como hemos explicado anteriormente, si el usuario no se ha identificado (registrado o conectado) no se puede realizar esta función. Se muestra un error de que el usuario no se ha identificado.

28. **Hacer un LIST_CONTENT sin estar conectado:** se muestra un mensaje de error.

29. **Hacer un LIST_CONTENT de un usuario que no exista:** se muestra un mensaje de que el usuario remoto no existe.

30. **Hacer un LIST_CONTENT correctamente:** se muestran todos los archivos publicados de un usuario.

31. **Hacer GET_FILE sin estar registrado (sin estar identificado):** como hemos explicado anteriormente, si el usuario no se ha identificado (registrado o conectado) no se puede realizar esta función. Se muestra un error de que el usuario no se ha identificado.

32. **Hacer GET_FILE sin estar conectado:** se muestra un mensaje de error. En esta función no se diferenciaba en el enunciado entre errores, así que se en la mayoría de los casos se muestra el mismo mensaje de error.

33. **Hacer GET_FILE de un usuario remoto que no existe:** se muestra el mismo error que en la prueba anterior ya que para obtener el fichero de otro usuario ese usuario tiene que estar registrado.

34. **Hacer GET_FILE de un usuario remoto que no está conectado:** el usuario remoto también tiene que estar conectado. Si el usuario remoto no está conectado no puede recibir peticiones de otros usuarios, por lo que la función devuelve un error.

35. **Hacer GET_FILE de una fichero que no existe:** en el caso de que se intente obtener un fichero que no exista la función devuelve un error específico y no se copia ningún fichero.

36. **Hacer GET_FILE correctamente:** cuando se cumplen todas las condiciones necesarias funciona correctamente. Si en el directorio del usuario que realiza la función ya existe el archivo donde se quiere copiar el nuevo archivo, este se sobrescribe. En caso contrario se crea y se copia su contenido.

CONCLUSIONES

Los principales problemas que hemos encontrado en la práctica han sido en la comunicación entre el cliente de python y el servidor principal de C. En cuanto entendimos como se mandaban los mensajes fue fácil de solucionar. También había aspectos que no nos quedaron claros en el enunciado y tomamos las decisiones de diseño explicadas anteriormente.

Consideramos que esta práctica ha servido para entender la mayor parte de la asignatura ya que cubre la mayor parte del temario. Lo hemos podido probar en máquinas distintas y el código seguía funcionando que es algo que no habíamos hecho en otras asignaturas. Hemos tenido varias dudas en algunos aspectos que no quedaban claros en el enunciado y que se podrían haber especificado mejor.