Redes de Computadoras

Curso 2022

Grupo 58

Informe - Obligatorio 2

Autores:
Emiliano Silva
Fabricio Techera
Joaquín Vidal

Supervisor:
Leonardo Vidal

Octubre 16, 2022

Contents

1 Solución planteada **2**

1.1 Cliente 2

1.2 Servidor 3

1.2.1 Función broadcast 3

1.2.2 Función discovery 3

1.2.3 Función start 4

1.2.4 Función re_alloc_info 4

1.2.5 Función handle_client 4

1.2.6 Función handle_request 4

1.2.7 Función check_message 4

1.2.8 Función handle_server 5

2 Problemas encontrados **6**

3 Posibles mejoras **7**

1 Solución planteada

La solución presentada consta de 2 archivos python. Uno llamado server.py, que implementa la lógica de un servidor, y otro, cliente.py que implementa la lógica de un cliente.

1.1 Cliente

Primeramente el cliente obtiene los argumentos pasados en el momento de su invocación. De aquí se obtienen la dirección del servidor, el puerto en el que escucha el servidor, la primera operación a realizar, la clave, y de ser necesario el valor a setear.

Para conectarse, el cliente crea un socket con la función socket(). Pasando como parámetros AF_INET que indica que el protocolo será IPv4 y SOCK_STREAM indica que la comunicación será TCP.

Con el par IP del servidor y puerto del servidor se crea la dirección del servidor, a la que se conectará el cliente.

En nuestro caso, el cliente llevará a cabo la operación indicada en el momento de la operación. Luego, leerá comandos de la consola y ejecutará lo solicitado. Los comandos son de la forma <Op> <Key> [<Value>].

- Op es la operación a realizar, que puede ser GET, SET o DEL.
- Key es la clave del valor al que se le desea aplicar la operación.
- Value solo es necesario en el caso de que la operación sea SET para indicar el valor a almacenar.

Una vez obtenido el comando es codificado a utf-8 y luego enviado al servidor. Una vez enviado, el cliente espera la respuesta del servidor. Esto se hace con la función recv(), que recibe como parametro el largo máximo de mensaje que recibirá.

Esta respuesta es decodificada e imprimida en pantalla para ser leida por el usuario.

En caso de desear finalizar la ejecución del cliente es necesario ingresar el comando EXIT, de esta forma se cierra el socket creado, finalizando así la transmisión.

1.2 Servidor

El servidor debe manejar varias cosas a la vez, para esto utilizamos la librería threading incluida en python. Esto nos permite manejar distintas funciones en hilos distintos, que se ejecutan concurrentemente. Particularmente, decidimos mantener un hilo para el broadcast, otro para descubrir otros servidores que se anuncien, y otro para cada cliente que se conecte al servidor.

1.2.1 Función broadcast

Esta función es la encargada de anunciar al servidor para los otros servidores, conectados a la red local. Para comunicarse con todos los servidores conectados, se utiliza la ip FF-FF-FF-FF-FF-FF, o 255.255.255.255. Esta es una ip reservada para broadcast, al utilizarla el mensaje será recibido por todos los servidores en la red local.

1.2.2 Función discovery

Es la función encargada de descubrir los otros servidores que son anunciados en la función previamente descripta.

Se crea un socket, utilizando la función socket de la librería socket. Con la función bind se relaciona al servidor actual con la dirección en la que escuchará.

Primero se lee del socket UDP, obteniendo lo recibido (cuando lo haya) y la dirección del emisor. En caso de que el mensaje recibido sea un announce y que la ip recibida sea distinta de la del servidor actual es necesario revisar que el nuevo servidor no esté entre los previamente guardados. En este caso estamos seguros de que se descubrió un nuevo servidor.

Al darse este caso es necesario tanto actualizar la lista de servidores, como distribuir las claves en la nueva lista de servidores. Para modificar la lista de servidores es necesario adquirir el mutex de la misma, ya que sino podría suceder que el servidor esté almacenando un valor en la base al mismo tiempo, y de con el servidor que debe guardar el valor obteniendo así un servidor que puede ser incorrecto. Luego de actualizar la lista y redistribuir los datos, se libera el mutex de la lista.

1.2.3 Función start

Esta es la primera función en ser invocada, se invoca luego de crear los hilos para anunciarse y para descubrir nuevos hilos. Esta función lo que hace es atender a los nuevos clientes, aceptando su conexión y creando un nuevo hilo para la misma.

1.2.4 Función re_alloc_info

Es la función invocada al descubrir un nuevo servidor. Su función es enviar al nuevo servidor todas las claves que se encuentran en el servidor actual, pero que le corresponden al nuevo.

Para esto recorre todas las keys almacenadas en el servidor, calculando para cada una la distancia con la firma de cada servidor y comparándola con la distancia al actual. En caso de ser menor, o ser igual pero el nuevo ser menor al actual, se le envía el comando para que el mismo la almacene y se elimina de la base de datos del servidor actual.

1.2.5 Función handle_client

Es la función invocada al realizarse una nueva conexión con el servidor actual. Esta se ejecuta en un nuevo hilo, y es la encargada de diferenciar entre una conexión con un cliente o un servidor. Es importante notar que diferencia entre clientes y servidores con respecto al servidor actual, por lo que un servidor realizando consultas al actual será visto como un cliente.

1.2.6 Función handle_request

Es la función invocada al tener que manejar una request de un cliente. Maneja las consultas a la base de datos, agregando, actualizando, obteniendo y eliminando valores de la misma. Para manipular una clave obtiene el mutex de la misma, de esta forma se asegura que ninguna clave sea accedida a la vez por dos clientes distintos, dando así resultados inesperados.

1.2.7 Función check_message

Esta función chequea los comandos enviados por los clientes, revisando sus sintaxis. También implementa la lógica para decidir si la clave se encuentra en este servidor

o en otro.

1.2.8 Función handle_server

Es la función invocada cuando la clave solicitada al servidor no se encuentra en el mismo, sino que debe ser consultado a otro. Esta función hace que el servidor actual actúe como cliente con respecto al servidor que debería poseer la clave (puede no estar almacenada) y transmitir el mensaje obtenido al cliente original.

2 Problemas encontrados

Al comenzar a desarrollar la solución planteada nos encontramos con diversos problemas que tuvimos que superar.

El primero fue manejar correctamente la sintaxis y funcionamiento de python, mas particularmente de la librería sockets. Para esto recurrimos tanto a la documentación de la misma como a tutoriales online.

Luego, al pensar la solución entendimos que el servidor debía realizar varias tareas de forma simultánea, para esto utilizamos la librería threading, aplicando lo aprendido en la materia de Sistemas Operativos.

Relacionado con lo anterior, utilizamos Locks, también de la librería threading, para manejar adecuadamente el acceso concurrente a la base de datos. Ya que cada servidor maneja cada cliente en un hilo distinto, podría suceder que más de un cliente intente modificar un cierto valor de la base de datos de ese servidor. Para evitar problemas, utilizamos un Lock para cada clave de la base de datos. De esta forma evitamos que dos hilos accedan a la vez a la misma clave. Otra posible solución sería solo permitir que un hilo acceda a la base a la vez. Esta solución es más simple, pero por el contrario hace la base menos eficiente.

3 Posibles mejoras

Sea el caso en el que un servidor A necesita una clave de un servidor B, el A actuara como cliente con respecto al B. Esto también puede suceder de forma inversa, siendo el B cliente del A.

Entendemos que la mejor solución es que se utilice la misma conexión tanto para un caso como para el otro. Esto no sucede en la solución planteada. Actualmente lo que sucede es que se crean dos conexiones distintas, una para cada fin.

Esto sucede porque un servidor es abstracto de quién esta estableciendo la conexión como cliente es un cliente real, o un servidor que actúa como cliente.

Otra posible mejora es manejar el caso en el que se cierra un servidor. En ese caso habría que redistribuir la base nuevamente, sin considerar al servidor faltante.

References

[1] Socket library - https://docs.python.org/3/library/socket.html

[2] Threading library - https://docs.python.org/3/library/threading.html

[3] Python Socket tutorial - https://realpython.com/python-sockets/