PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 - Programación Avanzada
1^{er} semestre 2016

Actividad 15

Networking

Instrucciones

Ingenieros de La Red Social lo han contactado y están desesperados. Están alcanzando niveles estratoféricos de tráfico y la arquitectura de sus sistemas simplemente no está dando abasto. El DCC ha realizado un agudo diagnóstico de los problemas de infraestructura de La Red Social y han concluido que un Key-Value Store (KVS) es la solución a todos sus problemas. No obstante, se acerca el fin de semestre y todos están ocupados redactando pruebas, por lo que le han pedido ayuda a usted para desarrollar el requerimiento de La Red Social.

Un KVS[1] consiste en un motor de base de datos que asocia llaves con valores. Puedes imaginarlo como un diccionario de Python, dónde los valores sólo pueden ser texto o números.

La idea es desarrollar un servidor que implemente un KVS y a la vez desarrollar un cliente que se pueda conectar al KVS. El servidor debe permitir que varios clientes puedan estar conectados a la vez. Los clientes que se conecten correctamente al KVS serán capaces de enviar una serie de comandos al servidor, los que causaran algún efecto en el KVS (crear keys, setear keys, suscribirse a keys, etc.).

Comandos

- `SET key value`: Guarda en el servidor con la llave `key` el valor `value` entregado por el cliente. El valor entregado por el cliente puede ser un string o un número. El servidor retornará `OK` en caso de éxito o un string vacío en caso contrario.
- `GET key`: El servidor envía al cliente el valor almacenado bajo la llave `key`. En caso de que no exista un valor guardado con esa llave debe enviar `(nil)`.

Ejemplo KVS:
Esto es lo que ve el cliente

```
>>> SET bastian 33 #Se guarda el valor 33 en la key bastian
OK                 #Se guardo exitosamente la llave
>>> GET bastian    #Se pide el valor de la key bastian
33                 #Se entrega el valor de la llave
>>> GET antonio    #Se pide el valor de la key antonio
(nil)              #Como la llave no existe entrega (nil)
```

[1] Puedes encontrar más información sobre los KVS en `https://en.wikipedia.org/wiki/Key-value_database`.

BONUS

El KVS deberá aceptar instrucciones para implementar el paradigma de mensajería *publish-subscribe*[2]. Bajo este paradigma, los clientes se **suscribirán** a ciertos canales y serán capaces de **publicar** valores a dichos canales. Luego, el servidor, en este caso el KVS, se encargará de propagar la información publicada a todos los clientes que se encuentren suscritos a dicho canal. Para lograr lo anterior deberá implementar las siguientes instrucciones:

- `SUBSCRIBE foo bar`: A través de este comando el cliente se suscribirá a los canales `foo` y `bar`. Cabe destacar que el cliente es capaz de suscribirse a más de un canal a la vez, separando los nombres de éstos con espacios. Retorna `OK` en caso de éxito o un string vacío en caso contrario.
- `PUBLISH foo bar`: Se publica el mensaje `bar` en el canal `foo`. El servidor debe encargarse de propagar la información al resto de los clientes suscritos y retornará un entero correspondiente al número de clientes que recibieron la información.
- `UNSUBSCRIBE foo bar`: Análogo a `SUBSCRIBE`, a través de este comando el cliente eliminará la suscripción a los canales `foo` y `bar`. Nuevamente, cabe destacar que se pueden indicar múltiples canales separándolos con espacios. Retorna los mismos mensajes que `SUBSCRIBE`.

Ejemplo *publish-subscribe*:
Veremos el caso de dos clientes: Cliente 1 y Cliente 2. Primero estos dos clientes se subscribirán al mismo canal ÏIC2233"

```
Cliente 1
>>> SUBSCRIBE IIC2233            #El Cliente 1 se suscribe al canal IIC2233

Cliente 2
>>> SUBSCRIBE IIC2233 IIC2343    #El Cliente 2 se suscribe a los canales
                                 IIC2233 y IIC2343
```

Ahora, cliente 1 y cliente 2 pueden enviar mensajes por el canal IIC2233.

```
Cliente 1
>>> PUBLISH IIC2233 AC15    #El Cliente 1 env\'ia el mensaje AC15 por el canal IIC2233
1                           #Indica que solo 1 cliente recibi\'o el mensaje

Cliente 2
>>> AC15                    #El Cliente 2 recibe el mensaje enviado por el Cliente 1
>>>
```

En el caso de que uno de ellos envíe un mensaje, todos los suscritos en el canal deben recibirlo.

Requerimientos del servidor

- Debe desarrollar un servidor que implemente un KVS con todas las instrucciones mencionadas en la tabla de comandos.
- Cada vez que el servidor acepte la conexión de un nuevo cliente deberá imprimir en la terminal el evento junto a un identificador único asignado a dicho cliente.
- Cada vez que se reciba algún comando desde alguno de los clientes, el servidor deberá imprimir el comando en la terminal, junto con la respuesta e identificador del cliente correspondiente.

[2]Puedes encontrar más información sobre este paradigma en `https://en.wikipedia.org/wiki/Publish-subscribe_pattern`.

- Debido a que muchos clientes accederán a recursos compartidos, debe implementar todos los *locks* necesarios para asegurar una ejecución consistente del sistema.

Requerimientos del cliente

- Desarrollar un cliente que pueda conectarse a un servidor. Cuando la conexión se ha completado debe avisar de esto en la terminal.
- Implementar un método que permita enviar comandos al servidor.

Notas

- Pese a que toda la interacción del usuario con el cliente debe ser a través de la terminal, ésta debe ser lo más natural y usable posible. Interfaces mal hechas (poco intuitivas, que fallen con inputs incorrectos del usuario, etc.) serán castigadas.
- Debe crear al menos dos archivos por separado: `servidor.py` y `cliente.py`.
- No es necesario que los datos persistan entre sesiones.

To - DO

- Crear clase Servidor (4.0 pts)
  - (2.0 pts) Manejo de conexiones del servidor (permite tener conectados de forma eficiente a varios clientes en paralelo, implementa todos los *locks* necesarios).
  - (1.0 pts) `SET key value` implementado correctamente.
  - (1.0 pts) `GET key` implementado correctamente.
- Crear clase Cliente (2.0 pts)
  - (1.0 pt) Iniciar conexión con el servidor.
  - (1.0 pt) Interfaz usable, capaz de enviar y recibir datos al servidor.
- Bonus: Agregar al servidor (3.0 pts)
  - (1.0 pts) `SUBSCRIBE foo bar` implementado correctamente.
  - (1.0 pts) `PUBLISH foo bar` implementado correctamente.
  - (1.0 pts) `UNSUBSCRIBE foo bar` implementado correctamente.

Entrega

- **Lugar:** GIT - Carpeta: Actividades/AC15
- **Hora:** 16:55