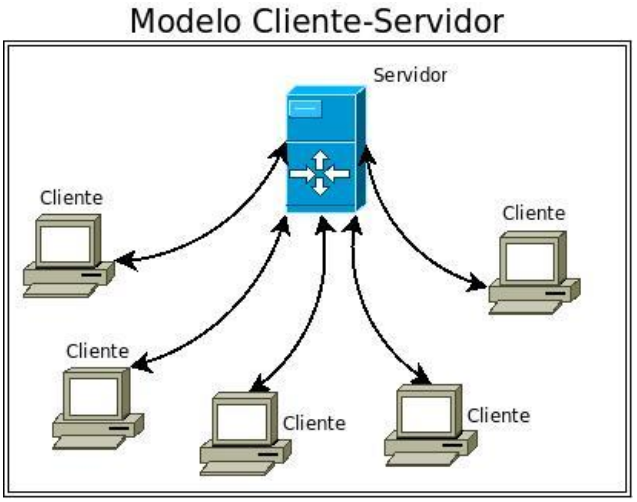
Trabajo de Estructuras de Control. Enunciado. Entrega Viernes 10 mayo.

Vamos a realizar una aplicación cliente-servidor que simule la red social Twitter. Como es lógico, realizaremos muchas simplificaciones al funcionamiento para que el trabajo sea abordable.



Estructura general:

- Aplicación cliente.py que tendrá el código del cliente. (incluyendo GUI)
- Aplicación servidor.py (sin GUI, solamente mostrará por terminal mensajes de control para conocer qué ocurre en el servidor)
- Para facilitar la aplicación utilizaremos Pandas como estructura de datos, aprovechando que ya tenemos cierto dominio de sus funciones para consultar tablas y para leer/escribir de la base de datos mediante pandas.read_sql(), lo que nos ahorrará mucho código y tiempo. Para estructurar el código en el servidor, las funciones que utilizan la base de datos y el DataFrame se realizarán en un archivo independiente del servidor llamado admindata.py.

Aparte de la librería Pandas, utilizaremos:

- Para el entorno gráfico del cliente: PySimpleGUI
- Para la comunicación cliente-servidor utilizaremos RPC, la librería gRPC.

Haremos la aplicación por partes, de manera que todos puedan llegar a una nota mínima aunque no se complete. La nota de cada parte es independiente:

1. (6 puntos) Funcionamiento básico del cliente-servidor sin entorno gráfico con la siguiente funcionalidad implementada tanto en cliente como en servidor:
    a. Registrarse como usuario (1p)
    b. Hacer login y logout (1p)
    c. Ver usuarios registrados (1p)
    d. Seguir a un nuevo usuario (1p)
    e. Ver usuarios a los que sigo (1p)
    f. Enviar tuits al servidor (1p)
2. (2 puntos). Añadir las funciones a admindata.py e implementarlas en cliente y servidor:
    a. Dejar de seguir a un usuario
    b. Recibir Tuits
3. (2 puntos) Añadir Interfaz gráfica al cliente con PySimpleGUI

No recomiendo meter interfaz gráfica desde el principio ya que nos resultará difícil saber qué está funcionando mal, si la interfaz , la lógica interna en el cliente o servidor. Una vez tenemos nuestra aplicación funcionando y bien probada, duplicamos el cliente.py en cliente_gui.py y procedemos a meter la interfaz gráfica.

Además de la funcionalidad se debe incluir, en servidor y cliente, pero sobre todo en el cliente, control de excepciones que consideréis oportuno. Si se produce una excepción en el servidor cuando se ejecuta un método remoto, la excepción se redirige al cliente. gRPC lanza excepción propia si el servidor no está online.

Se facilita lo siguiente:

- El fichero admindata.py con 2 clases que manejan los usuarios registrados por un lado y, por otro, los usuarios logueados y los mensajes enviados al servidor. Todo se almacena en forma de dataframes de pandas.
- El fichero proto con las librerías compiladas. Ni el fichero proto ni las librerías compiladas a partir de él deben ser modificadas para que todos sigamos la misma estructura de llamadas remotas.
- La estructura del cliente.py y del servidor.py con menús que se deben respetar y todas las cabeceras de las funciones a implementar junto a comentarios orientativos.

Funcionamiento de gRPC:

- El servidor expone varios métodos remotos que pueden ser llamados por los clientes.
- Cada método remoto tiene un mensaje de entrada (request) y un mensaje de retorno (reply)
- Estos métodos que expone el servidor se definen en un fichero protobuf con extensión .proto. Cada método en el fichero .proto debe tener un parámetro (Request) y un return (Reply).

```
rpc Login (LoginRequest) returns (LoginReply) {}
```

- Cada parámetro Request y Reply es un mensaje. La estructura de estos mensajes se definen en el mismo fichero. Un mensaje consta de una serie de campos, especificando el tipo de dato de cada campo. Por ejemplo, **LoginRequest** es el mensaje necesario para hacer el login. Sería

```
message LoginRequest {
        string user = 1;
        string password = 2;
}
```

El número no es un valor inicial, sino el orden del campo dentro del mensaje.

- El LoginReply es el mensaje de retorno del procedimiento Login. El servidor responde según se haya logueado con éxito o no. En este caso se retorna.

```
message LoginReply {
        int32 error = 1;
        int32 session = 2;
}
```

error = 0, no error, logueado correctamente.
error = 1, usuario incorrecto
error = 2, contraseña incorrecta

session es un identificador de sesión. Será un código numérico de 4 cifras único para cada sesión activa.

- Este fichero protobuf se “compila” con un comando y genera l2 ibrerías python que debemos usar en nuestro código para:
  - En el servidor: implementar los procedimientos remotos definidos en el .proto.
  - En el cliente: invocar los procedimientos remotos definidos en el .proto.

- En el lado cliente se crea un objeto “stub” a partir de las clases generadas por gRPC y desde ahí podemos llamar a los métodos del servicio definidos en el proto.