Sistemas Distribuidos.
Ejercicio Evaluable 2: Sockets TCP

Se desea diseñar e implementar el mismo servicio realizado en el ejercicio evaluable 1, pero en este caso utilizando sockets TCP. El servicio permite almacenar **tuplas <clave-valor1-valor2-valor3>**. La clave será un valor de tipo entero y los valores asociados a la clave será una **cadena de caracteres** de cómo mucho 255 caracteres (excluido el código 0 que permite indicar el fin de la cadena), un número entero (**int**) y un número **float**. Los clientes del sistema que hacen uso de este servicio deben tener disponible el mismo API que se definió para el ejercicio 1:

- `int` **`init`**`()`. Esta llamada permite inicializar el servicio de elementos clave-valor1-valor2-valor3. Mediante este servicio se destruyen todas las tuplas que estuvieran almacenadas previamente. La función devuelve 0 en caso de éxito y -1 en caso de error.
- `int` **`set_value`**`(int key, char *value1, int value2, float value3)`. Este servicio inserta el elemento <`key, value1, value2, value3`>. El servicio devuelve 0 si se insertó con éxito y -1 en caso de error. Se considera error, intentar insertar una clave que ya existe previamente. En este caso se indicará el error y no se insertará.
- `int` **`get_value`**`(int key, char *value1, int *value2, float *value3)`. Este servicio permite obtener los valores asociados a la clave `key`. Los valores se devuelven en `value1, value2` y value3. La función devuelve 0 en caso de éxito y -1 en caso de error, por ejemplo, si no existe un elemento con dicha clave.
- `int` **`modify_value`**`(int key, char *value1, int value2, float value3)`. Este servicio permite modificar los valores asociados a la clave `key`. La función devuelve 0 en caso de éxito y -1 en caso de error, por ejemplo, si no existe un elemento con dicha clave.
- `int` **`delete_key`**`(int key)`. Este servicio permite borrar el elemento cuya clave es `key`. La función devuelve 0 en caso de éxito y -1 en caso de error. En caso de que la clave no exista también se devuelve -1.
- `int` **`exist`**`(int key)`. Este servicio permite determinar si existe un elemento con clave `key`. La función devuelve 1 en caso de que exista y 0 en caso de que no exista. En caso de error se devuelve -1. Un error puede ocurrir en este caso por un problema en las comunicaciones.
- `int` **`num_items`**`()`. Este servicio devuelve el número de elementos almacenados en el servidor. La llamada devuelve -1 en caso de error.

Tenga en cuenta para las llamadas anteriores que también se considera error, por ejemplo, que se produzca un error en el sistema de paso de mensajes. En este caso, error en la conexión con el servidor, etc..

Todas las operaciones anteriores deben realizarse de forma **atómica** en el servidor.

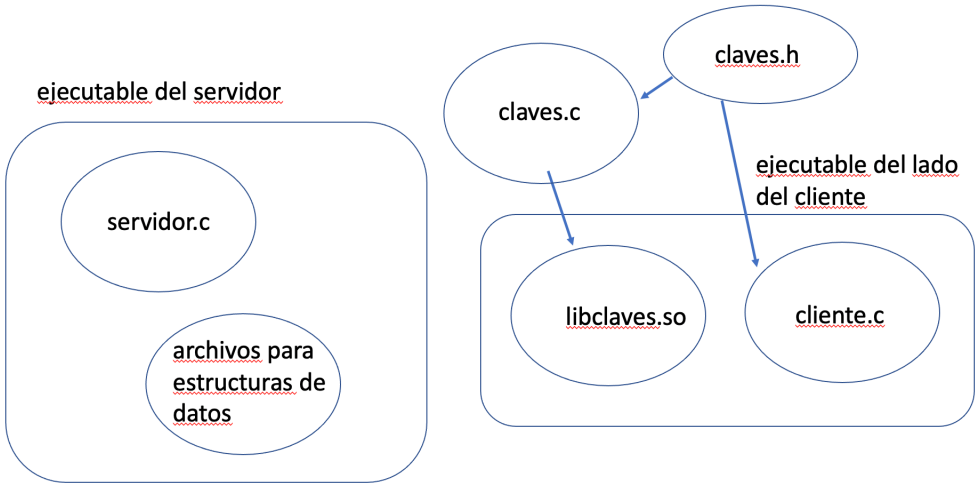
Diseñe e implemente, utilizando exclusivamente el **lenguaje de programación** C y **sockets TCP**, el sistema que implemente este servicio de elementos clave-valor1-valor2. Para ello debe:

1. Diseñar el protocolo de servicio que permite comunicar a los clientes con los servidores. El protocolo elegido tiene que ser independiente del lenguaje de programación, sistema operativo y hardware donde ejecuten cliente y servidor.l
2. En función del diseño realizado, desarrollar el código del servidor (**`servidor.c`**) encargado de gestionar las estructuras de datos que almacenan los elementos clave-valor1-valor2. Puede elegirse la estructura de datos que se estime oportuno, siempre que no imponga un límite en el número de elementos que se pueden almacenar. El servidor desarrollado debe ser **concurrente**.
3. Desarrollar, en función del diseño realizado, el código que implementa los servicios anteriores (**`init`**, **`set_value`**, **`get_value`**, **`delete_key`**, **`modify_value`**, **`exist`**, y **`num-items`**). El

código se desarrollará sobre el archivo con nombre **`claves.c.`** Este es el código que ofrece la interfaz a los clientes y se encarga de implementar los servicios anteriores (del lado del cliente) contactando con el servidor anterior. A partir de dicha implementación se deberá crear una biblioteca dinámica denominada `libclaves.so`. Esta será la biblioteca que utilizarán las aplicaciones de usuario que para usar el servicio. Debe investigar y buscar la forma de crear dicha biblioteca.

4. Desarrollar un ejemplo de código de un cliente (**`cliente.c`**) que utilice las funciones anteriores. El ejecutable de este programa tiene que generarse empleando la biblioteca desarrollada en el apartado anterior, es decir, el código de este cliente debe enlazarse con la biblioteca dinámica anterior. Este cliente se utilizará para probar el servicio desarrollado y deberá realizar las invocaciones al API de tuplas que considere oportuno. El código incluido en **`cliente.c`** solo podrá incluir llamadas a los servicios implementados y descritos anteriormente. En él no puede haber ninguna referencia a sockets. Para este cliente debe poder utilizarse el cliente desarrollado en el ejercicio 1, sin necesidad de ningún cambio.
5. Elaborar un plan de pruebas del servicio desarrollado. Este plan se probará con el código desarrollado en el apartado anterior.

La estructura del código a desarrollar se muestra en la siguiente figura:



Para este ejercicio, las partes del ejercicio 1 que hay que cambiar son el archivo **`claves.c`** **y** el **`servidor.c.`** tanto el cliente o clientes de pruebas utilizados en el ejercicio 1 como los archivos con la implementación del servicio de tuplas en el servidor no requieren cambios. Solo es necesario enlazar el cliente del ejercicio 1 con la nueva biblioteca desarrollada en este.

Material a entregar: Se deberá entregar la siguiente documentación:

Fichero **ejercicio_evaluable2.tar**, que incluirá, al menos:

- El código del cliente (`cliente.c`) donde se encuentran exclusivamente el uso de las llamadas al API que permiten probar el servicio desarrollado. Se pueden enviar diversos archivos cliente en función del plan de pruebas elegido. Solo habrá de documentarse correctamente en la memoria del ejercicio.

- El código del servidor (`servidor.c`) y el código correspondiente a la implementación de la biblioteca del lado del cliente (`claves.c`).
- Un fichero `Makefile`, que permite compilar todos los archivos anteriores y generar la biblioteca `libclaves.so`. Este `Makefile` debe generar además dos ejecutables: el ejecutable del servidor, que implementa el servicio y el ejecutable de cliente, obtenido a partir del archivo `cliente.c` y la biblioteca `libclaves.so`.
- Una pequeña memoria en **PDF** (no más de ocho páginas, incluida la portada), indicando el diseño realizado y la forma de compilar y generar el ejecutable del cliente y del servidor.
- Dentro del fichero comprimido `ejercicio_evaluable1.tar`, también se incluirán todos aquellos archivos adicionales que necesite para el desarrollo del servicio. Por ejemplo, ficheros que gestionan las estructuras de datos elegidas, etc.

Para el almacenamiento de los elementos clave-valor1-valor2-valor3 puede hacer uso de la estructura de datos o mecanismo de almacenamiento que considere más adecuado, el cual describirá en la memoria entregada. La estructura elegida no debe fijar un límite en el número de elementos que se pueden almacenar. Puede utilizarse la misma implementación realizada en el ejercicio 1.

El punto de partida para el desarrollo de este ejercicio es el ejercicio 1 y el código desarrollado en el laboratorio 2 dedicado a los sockets.

Se recomienda probar el servidor con varios clientes de forma concurrente.

Aclaraciones adicionales:

En este ejercicio la biblioteca a desarrollar tiene que conocer la dirección IP y el puerto del servidor que ofrece el servicio de tuplas. Para evitar tener una dirección físicamente programada en el código y no tener que pasar la IP y el puerto en la línea de mandatos del programa cliente, se va a considerar que tanto la dirección IP como el puerto se van a pasar utilizando dos variables de entorno:

- `IP_TUPLAS`: variable de entorno que define la IP del servidor.
- `PORT_TUPLAS`: variable de entorno que define el puerto del servidor de tuplas.

Estas variables de entorno habrán de definirse en cada terminal donde se ejecute el cliente. Para definir una variable de entorno denominada VAR1 con valor "`hola`" habrá que especificar en la consola:

```
$> export VAR1=hola
```

A continuación, se muestra un ejemplo de programa que es capaz de acceder al valor de dicha variable de entorno. Tenga en cuenta que la variable de entorno se devuelve como una cadena de caracteres.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
        char *var;

        var = getenv("VAR1");
        if (var == NULL){
                printf("Variable VAR1 no definida\n");
                return 0;
```

```
        }
        else
                printf("Variable VAR1 definida con valor %s\n", var);

        return 0;
}
```

Para conocer la lista de variables de entorno definidas es necesario ejecutar en la consola:

```
$> env
```

Para poder ejecutar el cliente será necesario definir las variables de entorno `IP_TUPLAS` y `PORT_TUPLAS` utilizando el comando `export` indicado anteriormente. Una forma alternativa de pasar los valores de estas dos variables de entorno al programa cliente es ejecutando este programa de la siguiente forma:

```
$> env IP_TUPLAS=localhost PORT_TUPLAS=8080 ./cliente
```

En cuanto al servidor, el puerto se le pasará como un argumento en la línea de mandatos de la siguiente forma:

```
./servidor <PUERTO>
```

Por ejemplo:

```
./servidor 4500
```

Especificará que el servidor aceptará peticiones de los clientes en el puerto 4500.