Sistemas Distribuidos.
Ejercicio Evaluable 3: RPC

El objetivo de este ejercicio es desarrollar el mismo servicio de los ejercicios 1 y 2 pero en este caso utilizando llamadas a procedimientos remotos (RPC). Para ello se utilizará el modelo ONC RPC visto en clase.

Debe **mantenerse** obligatoriamente la misma interfaz para los clientes:

- `int` **`init`**`(void)`. Esta llamada permite inicializar el servicio de elementos clave-valor1-valor2. 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 N_value2, double *V_value_2)`. Este servicio inserta el elemento <`key, value1, value2`>. El vector correspondiente al valor 2 vendrá dado por la dimensión del vector (N_Value2) y el vector en si (V_value2). 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 o que el valor N_value2 esté fuera de rango. En este caso se devolverá -1 y no se insertará. También se considerará error cualquier error en las comunicaciones.
- `int` **`get_value`**`(int key, char *value1, int *N_value2, double *V_value2)`. Este servicio permite obtener los valores asociados a la clave `key`. La cadena de caracteres asociada se devuelve en value1. En N_Value2 se devuelve la dimensión del vector asociado al valor 2 y en V_value2 las componentes del vector. Tanto value1 como V_value2 tienen que tener espacio reservado para poder almacenar el máximo número de elementos posible (256 en el caso de la cadena de caracteres y 32 en el caso del vector de doubles). La función devuelve 0 en caso de éxito y -1 en caso de error, por ejemplo, si no existe un elemento con dicha clave o si se produce un error de comunicaciones.
- `int` **`modify_value`**`(int key, char *value1, int N_value2, double *V_value2)`. 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 o si se produce un error en las comunicaciones. También se devolverá -1 si el valor N_value2 está fuera de rango.
- `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.

Tenga en cuenta, para las llamadas anteriores, que también se considera error, por ejemplo, que se produzca un error en el sistema de RPCs.

Diseñe e implemente, utilizando exclusivamente el **lenguaje de programación** C y **el modelo de ONC RPC**, el sistema que implementa este servicio. Para ello debe:

1. Diseñar y especificar la interfaz de servicio (archivo con extensión `.x`) que permite dar soporte a esta funcionalidad.
2. En función de la interfaz especificada, 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 asegurar una posible ejecución **concurrente** de los servicios.

3. Desarrollar, en función de la interfaz especificada, el código que implementa los servicios anteriores. 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`**) o varios, que utilice las funciones anteriores. El ejecutable/ejecutables 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 servicios de RPC.
5. Elaborar un plan de pruebas del servicio desarrollado. Este plan se probará con el código desarrollado en el apartado anterior. Se puede utilizar el mismo plan de pruebas de los ejercicios anteriores.

Un espeto importante a tener en cuenta en el desarrollo de este ejercicio es descubrir cómo integrar los archivos que se generan con el programa `rpcgen` para dar soporte a la funcionalidad descrita anteriormente. Por tanto, forma parte de este ejercicio el obtener la forma de integrar estos archivos en la solución final.

Material a entregar: Se deberá entregar el siguiente material:

Fichero **ejercicio_evaluable3.tar**, que incluirá, además de los archivos necesarios para la compilación del cliente y del servidor, los siguientes elementos:

- Un archivo `Makefile`, que permite compilar todos los archivos necesarios 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 o clientes utilizados para probar el sistema, obtenido a partir del archivo `cliente.c` (u otros clientes de prueba) y la biblioteca `libclaves.so`. En este `Makefile` se tendrán que incluir aquellos ficheros obtenidos con `rpcgen` que sean necesarios.
- Una pequeña memoria en **PDF** (no más de ocho páginas, incluida la portada), indicando el diseño realizado, la estructura de archivos utilizado en la compilación del servidor, de la biblioteca `libclaves.so` y del cliente y la forma de compilar y generar el ejecutable del cliente y del servidor.
- Dentro del fichero comprimido `ejercicio_evaluable3.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 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. Utilice la misma implementación realizada en el ejercicio 1 y 2.

El punto de partida para el desarrollo de este ejercicio son los ejercicios 1 y 2.

En este ejercicio el cliente y la biblioteca a desarrollar tiene que conocer la dirección IP del servidor que ofrece el servicio de tuplas (para las RPC no es necesario conocer el puerto). Para evitar tener una dirección físicamente programada en el código y no tener que pasar la IP en la línea de mandatos del programa cliente, se va a considerar que la dirección IP se va a pasar utilizando la siguiente variable de entorno:

- `IP_TUPLAS`: variable de entorno que define la IP donde ejecuta el servidor RPC.

Esta variable de entorno habrá de definirse en cada terminal donde se ejecute el cliente.

En cuanto al servidor, en este caso no será necesario pasarle en la línea de argumentos ningún puerto y este se ejecutará de la siguiente forma:

```
./servidor
```

La entrega se realizará mediante Aula Global en el entregador habilitado. La fecha límite de entrega es: 21/04/2024. (23:55 horas).