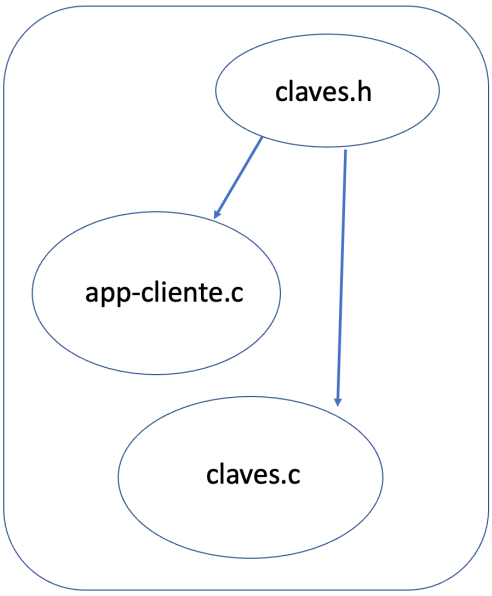
Sistemas Distribuidos. Curso 2024-2025
Ejercicio Evaluable 1: colas de mensajes

Considere una aplicación monolítica con la siguiente estructura:



El archivo `claves.c` implementa un servicio para almacenar tuplas de la forma **<key-value1-value2-value3>**, donde la clave será un número entero (**int**) y los valores asociados a la clave serán:

- value1: una **cadena de caracteres** de como mucho 255 caracteres (excluido el código 0 que permite indicar el fin de la cadena).
- value2: un **vector** de N elementos de tipo **double** (coma flotante de doble precisión). El valor 2 vendrá dado por dos elementos: un número N que determina la longitud del vector, y el vector con los elementos. El valor de N será cualquier número comprendido entre 1 y 32.
- value3: **una estructura** de datos que almacena dos valores enteros (x, y). Esta estructura se define de la siguiente forma:

```
struct Coord {
   int x;
   int y;
};
```

La API (*Application Progamming Interface*) de este servicio, cuyo prototipo en C se encuentra declarado en el archivo `claves.h`, incluye los siguientes servicios:

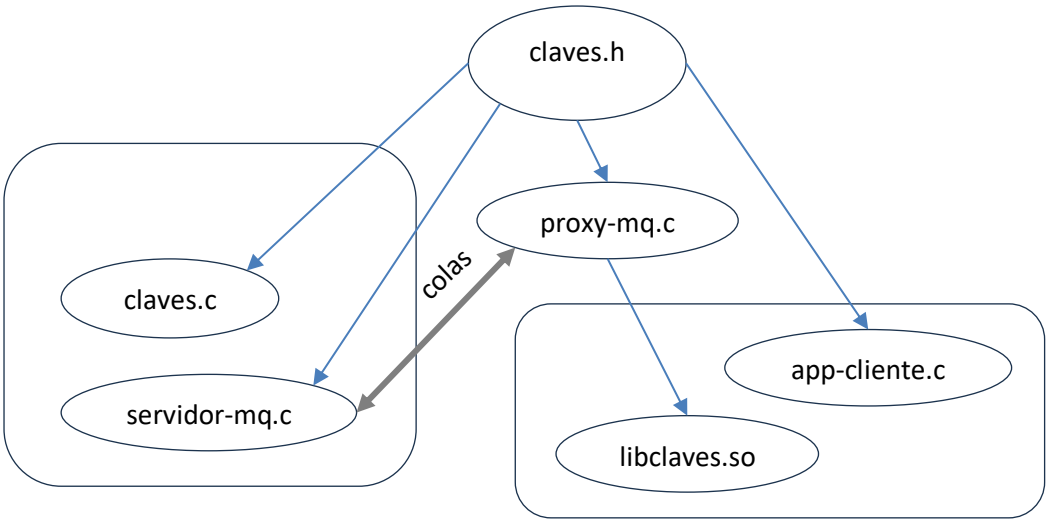
- `int` **`destroy`**`()`. Esta llamada permite inicializar el servicio de elementos `key-value1-value2-value3`. Mediante este servicio se destruyen todas las tuplas que estuvieran almacenadas previamente. La función devuelve 0 en caso de éxito y -1 en caso de error.
- `int` **`set_value`**`(int key, char *value1, int N_value2, double *V_value2, struct Coord value3)`. Este servicio inserta el elemento <`key, value1, value2, value3`>. El vector correspondiente al valor 2 vendrá dado por la dimensión del vector (`N_value2`) y el vector en sí (`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 (los valores tienen que estar comprendidos entre 1 y 32). En este caso se devolverá -1 y no se insertará.

- `int` **`get_value`**`(int key, char *value1, int *N_value2, double *V_value2, struct Coord *value3)`. 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` los componentes del vector. En `value3` se devuelve la estructura asociada. Tanto `value1` como `V_value2` tienen que tener espacio reservado para poder almacenar el máximo número de elementos posible (255 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.
- `int` **`modify_value`**`(int key, char *value1, int N_value2, double *V_value2, struct Coord 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. 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.

El archivo `app_cliente.c` es un ejemplo de aplicación que puede hacer uso de esta API. Un ejemplo posible sería este:

```
#include <stdio.h>
#include "claves.h"

int main (int argc, char **argv)
{
      int    key = 5;
      char  *v1 = "ejemplo de valor 1";
      double v2[] = {2.3, 0.5, 23.45};
      struct Coord v3;

      v3.x = 10;
      v3.y = 5;

      int err = set_value(key, v1, 3, v2, v3);
      if (err == -1) {
            printf("Error al insertar la tupla\n");
      }

      return 0;
}
```

En este ejercicio se trata de convertir esta aplicación en un servicio distribuido de forma que exista un servidor que gestione las claves y un cliente que tenga acceso a la API anterior, pero de forma distribuida. En este caso la aplicación tendrá la estructura de la siguiente figura:



En esta aplicación los ficheros `claves.h`, `claves.c` y `app-cliente.c` son los mismos que se tendrían en una aplicación monolítica. La interfaz de `claves.c` es idéntica a la anterior. Como en este caso existe una comunicación entre la parte cliente y la parte servidora, las funciones anteriores devolverán -1 cuando hay un error relacionado con el servicio de tuplas y -2 cuando hay un error producido por el servicio de comunicaciones. Es decir, las llamadas anteriores devolverán -2 cuando se produzca un error en el sistema de paso de mensajes (colas inexistentes, errores al enviar datos a una cola, errores al recibir datos de las colas, etc.). También se considerará error y se devolverá -1 sin necesidad de contactar con el servidor, en caso de que a las llamadas anteriores se pasen cadenas de más de 255 caracteres para el valor `value1` o cuando el valor `N_value2` sea mayor de 32.

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

Diseñe e implemente, **utilizando exclusivamente colas de mensajes POSIX** y el lenguaje de programación C, el sistema que implemente este servicio de elementos `key-value1-value2-value3`, con la estructura mostrada en la figura anterior. Para ello debe:

1. Desarrollar el código (**`claves.c`**) que gestiona el almacenamiento y procesamiento de las tuplas. Puede elegirse la estructura de datos que se estime oportuna, siempre que no imponga un límite en el número de elementos que se pueden almacenar.
2. Desarrollar el código del servidor (**`servidor-mq.c`**) encargado de las comunicaciones con la parte cliente. Este servidor **debe ser concurrente**, recibirá las peticiones del lado cliente e invocará a las funciones implementadas en `claves.c`.
3. Desarrollar el código que implementa el código (**`proxy-mq.c`**) de las llamadas anteriores (`destroy`, `set_value`, `get_value`, `delete_key`, `modify_value` y `exist`) en el lado del cliente. Este código suplanta a las funciones de la API y se encarga de realizar las peticiones al servidor. Es decir, 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 para usar el servicio. Debe investigar y buscar la forma de crear dicha biblioteca.

4. Desarrollar un ejemplo de código de un cliente (**app-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 a la API de tuplas que consideren oportunas. El código incluido en **app-cliente.c** solo podrá incluir llamadas a los servicios implementados y descritos anteriormente. En él no puede haber ninguna referencia a colas de mensajes.
5. Elaborar un plan de pruebas del servicio desarrollado. Este plan se probará con el código desarrollado en el apartado anterior, es decir en **app-cliente.c**. Se pueden entregar distintos ejemplos de aplicación cliente dependiendo del plan de pruebas elaborado.

Material a entregar

Se deberá entregar un fichero **ejercicio_evaluable1.zip**, que incluirá, al menos:

- El código del cliente (app-`cliente.c`) donde se encuentran exclusivamente el uso de las llamadas a la API que permiten probar el servicio desarrollado. En caso de que se entreguen varios programas clientes serán `app-cliente-1.c`, `app-cliente-2.c`, y así sucesivamente.
- El código del servidor (`servidor-mq.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 (o clientes si se entregan varios), obtenido a partir del archivo app-`cliente.c` y la biblioteca `libclaves.so`.
- Una pequeña memoria en **PDF** (no más de cinco 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.zip` también se incluirán todos aquellos archivos adicionales (.c o .h) que necesite para el desarrollo del servicio (como ficheros que gestionan las estructuras de datos elegidas, etc.). Por ejemplo, el archivo `claves.c` podría hacer uso de otros archivos adicionales que implementaran distintos aspectos de la aplicación.

Para el almacenamiento de los elementos `key-value1-value2-value3` puede hacer uso de la estructura de datos o mecanismo de almacenamiento que considere más adecuado (listas, ficheros, etc.), el cual describirá en la memoria entregada. La estructura elegida **no** debe fijar un límite en el número de elementos que se pueden almacenar.

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

Tenga en cuenta que el prototipo de las funciones (claves.h) `destroy`, `set_value`, `get_value`, `delete_key`, `modify_value` y `exist` no puede ser modificado.

La entrega se realizará mediante Aula Global en el entregador habilitado. La fecha límite de entrega es el 16/03/2025 a las 23:55 horas (horario marcado por el entregador de Aula Global).

Aclaraciones adicionales:

Cuando se esté desarrollando este ejercicio, es posible que en algunas ocasiones las colas de mensajes se queden en un estado inconsistente y tanto el cliente como el servidor fallen en tiempo de ejecución. En estos casos, lo mejor es borrar las colas de mensajes que se hayan quedado en el sistema. Para ello, pueden eliminarse las colas desde el directorio `/dev/mqueue`.