
# Universidad Carlos III
# Sistemas Distribuidos
# Curso 2024-25

# Ejercicio evaluable 1 - Colas de mensajes

Grupo 86
100451326 LÓPEZ NAVARRO, CÉSAR (100451326@alumnos.uc3m.es)
100451213 MARTÍNEZ CRUZ, ADRIÁN (100451213@alumnos.uc3m.es)

# Introducción

En este primer ejercicio evaluable se nos pide utilizar colas de mensajes POSIX para la creación de un servicio cliente-servidor en C. Para implementar este sistema, además se nos pide que implementemos las siguientes funciones:

- `int destroy()`
- `int set_value(int key, char *value1, int N_value2, double *V_value2, struct Coord value3)`
- `int get_value(int key, char *value1, int *N_value2, double *V_value2, struct Coord *value3)`
- `int modify_value(int key, char *value1, int N_value2, double *V_value2, struct Coord value3))`
- `int delete_key(int key)`
- `int exist(int key)`

La implementación de estas funciones se encuentra en claves.c

# Estructura de datos y funcionalidad real del programa

El servidor ejecutará una serie de servicios para el almacenamiento de tuplas con el siguiente formato:

<clave-valor1-valor2-valor3>

donde estos serán:

- Un número entero
- Una cadena de caracteres
- Un vector de N elementos de tipo `double`
- Una estructura de datos que almacena dos valores de tipo `int`

# Comunicación entre el servidor y el cliente

En este sistema distribuido, el cliente se comunica con el servidor mediante colas de mensajes POSIX. El cliente utiliza una biblioteca dinámica (`libclaves.so`) que actúa como proxy para enviar las solicitudes. El proxy crea una cola exclusiva para cada cliente, identificada por su PID, para recibir respuestas. Posteriormente, el proxy construye un mensaje que incluye el código de operación y los datos necesarios. A continuación, se envía el mensaje a la cola del servidor usando la función `mq_send`.
El servidor permanece a la espera de mensajes en su cola designada y los recibe con `mq_receive`. Una vez recibido el mensaje, el servidor interpreta el código de operación y ejecuta la función correspondiente. El resultado de la operación se almacena en el mismo mensaje tras su procesamiento.

Luego, el servidor abre la cola exclusiva del cliente para enviar la respuesta mediante `mq_send`. La respuesta se transmite de regreso al cliente a través de la cola que se creó específicamente para ello. El proxy del cliente recibe la respuesta utilizando `mq_receive` y extrae el resultado del mensaje.
Finalmente, la función en la biblioteca dinámica devuelve el resultado al programa cliente. Durante este proceso se emplean mecanismos que aseguran la atomicidad de cada operación. Asimismo, se gestionan adecuadamente los errores de comunicación mediante códigos de retorno. En conclusión, la comunicación se realiza de forma distribuida, eficiente y segura entre ambos componentes.

## Pruebas

Para la realización de las pruebas se ha decidido insertar, borrar y modificar, entre otras acciones, una serie de tuplas con diferentes valores de `key`. Con éstas, podremos tanto comprobar que todas las funciones se ejecutan correctamente y devuelven los resultados esperados como asegurar que las tuplas con valores incorrectos no sean introducidas.

- destroy(): Invocación para comprobar que los valores previamente almacenados en data.txt se borran correctamente.
- exist(): Llamadas a exist para tuplas con claves que han sido introducidas y claves que no.
- set_value(): Inserción de tuplas en las que se evalúa que se introducen datos correctos, y que se rechazan aquellas con parámetros inválidos o duplicados.
- get_value(): Se utiliza para recuperar los datos asociados a una clave insertada, comprobando que la función devuelve la cadena, el vector y la estructura de coordenadas correspondientes, y que notifica un error cuando la clave no existe.
- delete_key(): Permite eliminar una tupla, asegurándose de que, tras su invocación, la clave borrada ya no es reconocida por el sistema, así como de que se maneja apropiadamente el intento de borrar claves inexistentes.
- modify(): Se actualizan los valores asociados a una clave ya existente, comprobando que la modificación se refleja correctamente al recuperar la tupla actualizada y que se gestionan de forma adecuada los errores al intentar modificar una clave que no se encuentra en el almacenamiento.

## Compilar y ejecutar

Será necesario abrir dos terminales en linux o MAC.

En la primera habrá que ejecutar el comando

```C/C++
make re && ./servidor.out
```

Y en la segunda

```C/C++
make re && ./cliente.out
```

en ese orden.

En app-cliente.c viene implementado un posible ejemplo de un cliente como se explica anteriormente, por lo que tras ejecutar estos comandos, se verá el resultado del ejemplo en concreto.