
# Universidad Carlos III

2023-24

Ejercicio 3 - Sistemas Distribuidos

Pablo García Rius 100428976

José Antonio Barrientos Andrés 100451290

Curso 2023-24

# Diseño

## Servidor

### Estructura de datos

Se especifica en el enunciado del ejercicio que el único requisito para la selección de la estructura de datos es que no limite el número de claves que se puedan almacenar, por tanto, nuestra elección ha sido un árbol binario, en esta implementación del árbol binario, cada nodo está compuesto por la información de la tupla a almacenar, es decir, una key, la cadena de caracteres, la longitud N del vector y el vector de doubles. Además de punteros a los nodos izquierdo y derecho que salen de él.
La estructura utiliza como clave de ordenación la key, ya que es irrepetible, y hace como cualquier árbol binario, compara el valor de la key y si es menor buscará en el nodo izquierdo, si es mayor en el derecho, hasta encontrar un hueco vacío en el que colocar el nuevo.
Esta estructura de datos nos proporciona una cantidad ilimitada de nodos, además de una velocidad bastante considerable a la hora de hacer una búsqueda.

### Estructura del código

Se han declarado unas constantes que corresponden a los códigos de operación que el “cliente” puede mandar, estas son un entero que va del 0 al 5, siendo los valores: INIT, GET, SET, MODIFY, DELETE y EXIST.
Lo primero que hace el servidor es definir definir un estruct en el que se empaquetarán los argumentos de entrada y salida que mande el cliente, en este struct hay dos elementos, args_in y args_out que a su vez son dos struct ya definidos en servicio.x como los argumentos de entrada y salida del servicio de intercambio de mensajes con RPC.
args_in contiene la clave de operación, la key, la cadena de caracteres, la N y el vector de doubles, args_out contiene el resultado de la operación en code, la cadena de caracteres, la N y el vector de doubles que se devuelven al cliente.
El servidor hace un switch del código de operación recibido en args_in y manda a la función correspondiente el paquete con args_in y args_out, la función realiza la operación correspondiente y devuelve el código de resultado y los datos necesarios en args_out.

### Respuesta

En el atributo code de args_out se especifica el código de resultado 0 o -1 (o 1 en el caso de exist),la operación get se comporta de una forma un poco diferente, porque además, mandará el resto de argumentos en sus respectivos atributos en args_out.

El código en la respuesta es igual en las 4 primeras funciones: init, get, set, modify y delete devuelven 0 en caso de éxito y -1 en caso de error. Exist devuelve 0 en caso de no existir la tupla, 1 en caso de sí existir, -1 en caso de error.

# Claves

Para implementar **init** se realiza una llamada al procedimiento remoto que inicializa el servicio en el servidor

Para la función **set_value** envía todos los datos al servidor mediante un aestructura que agrupa todos los argumentos y espera la respuesta del servidor.

En la función **get_value** es parecida a set value, pero despues de enviar la clave, espera a recibir una estructura con todos los valores asociados.

La función **modify_value** es muy parecida a **set_value,** ya que envía todos los argumentos que se le pasan para que se pueden modificar.

En **delete_key** se envía el código de operación y la key a eliminar, después solo espera a que reciba la respuesta del servidor.

Para **exist** es igual que para delete, ya que solo envía la operación y la key y espera a la respuesta.

# Cliente

Para la parte de cliente hemos generado unos datos para insertar y probar las distintas funciones de las claves.

# Compilación

El fichero Makefile genera el ejecutable de clientes y de servidor utilizando el comando make en la línea de mandatos. Además creará la librería dinámica libclaves.so dentro del directorio lib del proyecto (Si este directorio no existe al momento de intentar compilar, sucederá un error)
Para poder utilizar la librería enlazada, hay que declarar varias variables de entorno en la terminal que se esté utilizando, de no hacerlo así, los ejecutables darán error al intentar ejecutarse.
Estos comandos son:

```
IP_TUPLAS=localhost
export IP_TUPLAS
LD_LIBRARY_PATH=$(pwd)/lib
export LD_LIBRARY_PATH
```

Para ejecutar cada programa, se hará de la siguiente manera:
Al estar utilizando RPC, no es necesario especificar al servidor la dirección en la que este se va a ejecutar.
Por tanto, para ejecutarlo solo será necesario hacer:
./servidor
Y de la misma forma para el cliente, ya que este extrae el hostname de la variable de entorno que hemos definido, solo será necesario hacer:
./cliente
Al ejecutar el fichero Makefile, se generan los ficheros necesarios para la comunicación con RPC, por tanto, no es necesario ningún paso adicional para conseguir esto.
Haciendo Make clean se eliminan los ficheros generados durante la compilación.