Sistemas Distribuidos

Ejercicio 1 - Colas de Mensajes

Marzo 2024

Ruben Castro Pruneda `100454441@alumnos.uc3m.es`

Alberto Molina Felipe `100454278@alumnos.uc3m.es`

Diseño

El sistema funciona como se especifica en el enunciado y como se muestra en el diagrama. El cliente utiliza las llamadas que ofrece el proxy y este se comunica con el servidor a través de colas de mensaje `POSIX`. Las siguientes han sido nuestras decisiones de diseño.

Mensajes

Los datos esenciales para el funcionamiento son los argumentos de la función. `value1`, , `N_value2`, `value2`, `value3`, `key` y un código para identificar el resultado de la operación.

Además de estas hemos añadido identificación del cliente para que el servidor conozca en que cola esperar. Para aceptar clientes concurrentes hemos incluido el `pid` y `tid` del proceso que llama a cualquiera de las funciones y con estos parámetros se identifican las colas de respuesta. Por último, enum flags comunican que tipo de comando se incluye en la petición.

Servidor

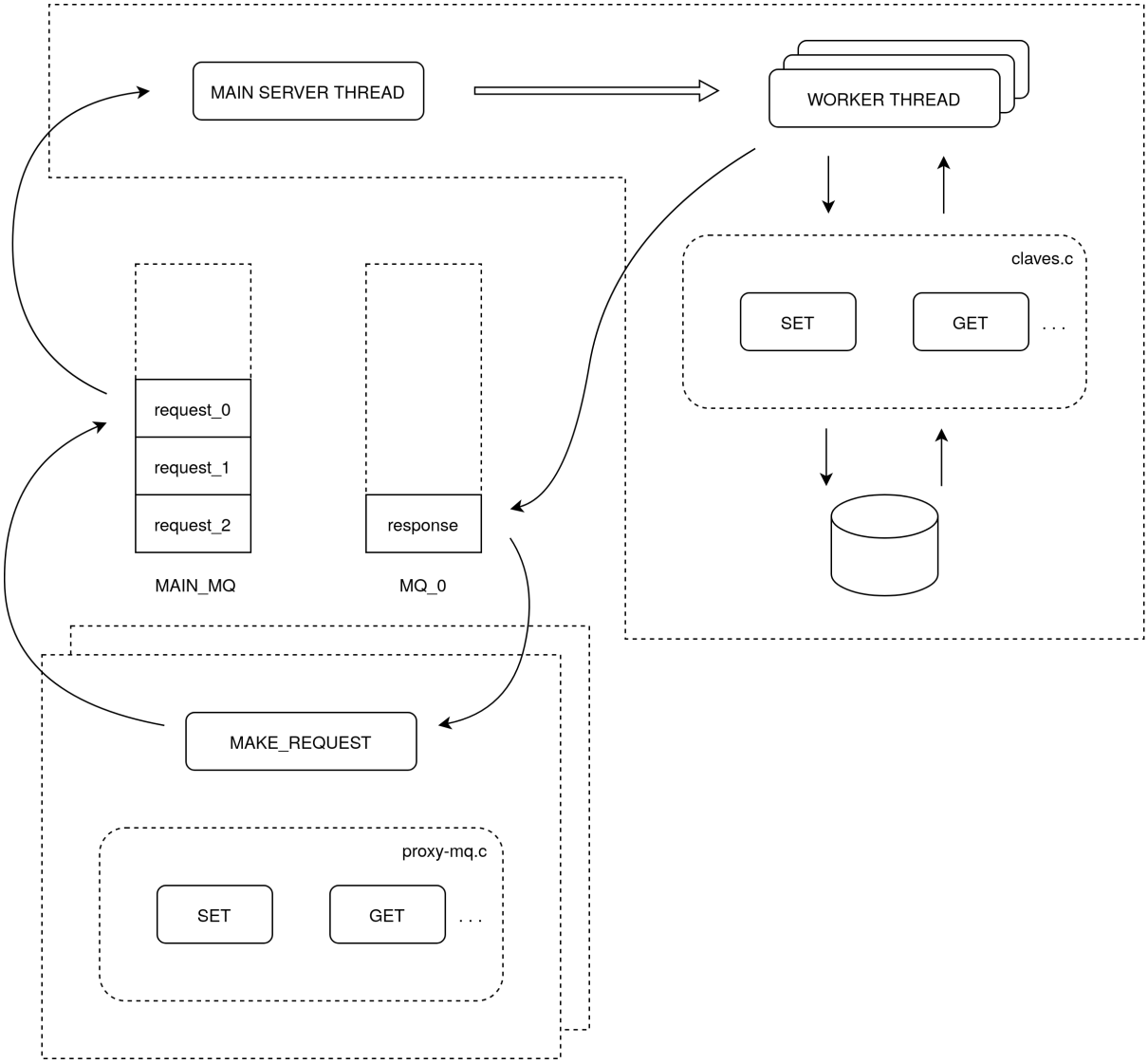
Almacenamiento

Hemos usado una simple lista enlazada para almacenar todos los pares clave-valor en el servidor. No es la opción más eficiente, pero sí la más sencilla y menos propensa a errores de implementación. Una tabla hash, en cambio, ofrecería un mejor rendimiento.

Concurrencia y mutex

Nuestro ejemplo de cliente usa un único hilo, pero el servidor si es concurrente. Un hilo principal se ocupa de leer los mensajes de la cola principal y crear *worker-threads* que ejecutan la lógica de negocio y responden a la consulta en la cola correspondiente.

Añadir concurrencia al servidor significa que la lista enlazada donde se almacenan los pares clave-valor necesita un mutex. Hemos decidido abstraer el mutex del servidor e incluirlo directamente en la implementación de `set`, `get`, ...



Tests

Nuestro cliente de ejemplo ejecuta una serie de pruebas para verificar el correcto funcionamiento del sistema de almacenamiento clave-valor. Las pruebas incluyen:

- Guardar un valor y recuperarlo para comprobar que coincide.
- Intentar obtener un valor de una clave que no ha sido almacenada.
- Intentar almacenar datos con un formato inválido para verificar que se detectan errores.
- Comprobar que no se puede sobrescribir una clave ya establecida.
- Verificar que una clave existe antes y después de eliminarla.
- Almacenar un valor, modificarlo y recuperarlo para confirmar los cambios.
- Intentar modificar una clave que no existe para comprobar que se genera un error.

```
$ ./bin/cliente --verbose
TEST set_get_success...
    SET: "Music City USA"
    GET: "Music City USA"
TEST get_doesnt_exist...
    GET: attempt to get non-existent key
TEST test_set_wrong_format...
    SET: N_value 40
    SET: String length 299
TEST set_already_existing_key...
    SET: first set
    SET: attempt to set existing key
TEST exist_delete_success...
    SET: "Paint It Blue"
    EXIST: before delete
    DELETE
    EXIST: after delete
TEST test_set_modify_get_success...
    SET: "Tennesse Special"
    MODIFY: "Honest Fight"
    GET: "Honest Fight"
TEST modify_doesnt_exist...
    MODIFY: attempting to modify a non-existent key
```

Compilación

Como indica el enunciado se compila el proxy como una biblioteca compartida y se generan dos binarios `bin/cliente` y `bin/servidor` con el comando `make`, ambos aceptan un flag para la verbosidad de los prints por pantalla.