uc3m | Universidad Carlos III de Madrid

BACHELOR'S DEGREE IN COMPUTER SCIENCE AND ENGINEERING

PRACTICAL EXERCISE 1

Students:

Moisés Hidalgo Gonzalez - 100405918

Álvaro Morata Hontanaya - 100405846

SYLLABUS

DESIGN

In this part of the document we will give a brief description of the different design choices we have made to develop this exercise.

DATA STRUCTURE

The data structure we have chosen is based on an external storage implementation. There will be a folder named "`/messages`" in the parent directory that will be created with the function `init()`. Inside this folder there will be files with the naming "`{key}.txt`". These files will be the messages of the database. Their contents will be a string with the format "`key-value1-value2-value3`", where:

- `key` is an integer that will identify the message.
- `value1` is a string of a maximum of 255 characters.
- `value2` is an integer.
- `value3` is a floating point number.

There will be only one message per file and vice versa.

SERVER

In the file "server.c" we have programmed everything related to our data structure, as well as the main loop of the server. It also contains the server-side implementation of the functions stated in the statement of the exercise. This server-side implementation means that these functions are called by the client to retrieve data or store it in the database. The function they execute is communicating directly with the files that store the messages and returning the data through a message queue.

This file also contains a method called "`deal_request()`" that processes the request sent by the client through the message queue, implemented with mutual exclusion in mind so that the server is concurrent and there is no race condition when accessing a file or variable.

For the last part of this file, we have the main loop of the server, implemented using multithreading and message queues to receive the requests from the client.

KEYS

Here we implemented a "`keys.h`" header which will store all the function headers requested in the practice assignment and will be accessed through the "`keys.c`" file.

For this we deal with two types of struct: request and response. Then, we make use of a "`send_request()`" method that will take as an input a "`request`" type of data (struct holding the data for the request) and return a "`response`" type of data (struct holding all the fields relative to the response). This method of "`send_request()`" will do all the message queue handling on the client side and will send the messages

that will be later received by the server with a specific operation code for each method and thus will process it accordingly. The individual methods are setting / receiving the data that we want to interact with the server side which will perform all operations atomically.

CLIENT

The file "`client.c`" includes the "`keys.h`" headers, accessing all of the methods programmed and required by the exercise.

This file contains different methods:
First, we have the "`sendData()`" method that will call the "`set_value()`" function with some predefined values and will send them to the server so it will store it in the backend as a file.
Next, we have the "`reset()`" method that will call the "`init()`" method, destroy all existent files in the "database" with it, and insert a new one with the "sendData()" method.

Then, we made unit tests for each of the methods in "`keys.c`" to later call them in a "`test_function()`" that will be run in the threads. This "`test_function()`" calls the different unit tests and "`reset()`" the data before each.

COMPILATION

The compilation of the project has been realized using CMake. In this section we will explain how to compile the project using this tool.

COMPILATION USING CMAKE

The file "`CMakeLists.txt`" has been written using the different instructions that allowed us to compile the source code and the shared library in the same way we were able to do with GCC.
To compile the different files we need to enter the following commands into a linux terminal being in the same folder as the file "`CMakeLists.txt`" is and the C files:

```
> cmake .  #It will create a makefile to compile the project
> make     #Will compile the files using gcc
```

With these two commands the project will be compiled, delivering us the executables "`server`", "`client`" and the dynamic library "`libkeys.so`".
To run the programs, simply use these two commands in different instances:

```
#Terminal 1:
> ./server
#Terminal 2:
> ./client
```

PROBLEMS FOUND & CONCLUSIONS

Firstly, the tests for `get_value()` and `exists()` fail. We have been testing the methods and trying to debug them, but no solution was found. It probably is something associated with pointers, or the contents passed to the struct response. The server-side part of the methods works well. They read the data from the files without a problem, but when the communication with "`keys.c`" is established and we try to retrieve the values returned by the method, they end up being not assigned back.

The other problem we have is related to multithreading testing. When we try to join two threads to the program, we get a segmentation fault exception, and we do not really know where the problem is. Testing for the other methods with a single thread works fine (exceptuating the 2 stated above). We tried to debug it exhaustively and narrowed it down but we couldn't find the exact spot where it fails.

We would like to have some feedback to identify what the problems are with the code to build upon this practice for the next exercises, to be able to learn and improve it. Our perspective right now is that it's a solid project in which we invested a lot of time learning and implementing. It evidently has some faults that are not making it work exactly as we want but could be easily polished with some guidance.