UNIVERSIDAD CARLOS III DE MADRID



https://lh5.googleusercontent.com/HWS1VKexp4khR52ZwzR5rmyeAb8AQyqxoKk_NMAxx4EJO0sieOw4ZxZkuZGtv24kOZJSE3oWRX9g55YoG7GVWWw5hMw73b4uyPSoAKCOIoqGJEnbt1xhNaHWm8TOyR7NdX07ndSenq-YnaiDQA

**DISTRIBUTED**

**SYSTEMS**

Exercise 1: message queues

**Francisco Colmenar**

**Lamas**

Description of the code

This lab assignment aims to simulate the implementation of a concurrent server. Moreover, this server provides some functionalities in order to interact with a database (which in our case is a list). The main goal of this lab is to provide a deep understanding of POSIX message queues applied to concurrent servers. The main roles of this implementation are the following ones:

* *server*: As it names indicates it is the program in charge of the implementation of the server. Furthermore, it is the one which interacts with the database as well as it receives the requests of the clients creating a new thread for each arrived request. Besides, its operations are atomically in order to ensure that there is no problem while accessing to the data.
* *doubleLinkedList*: It is a simple implementation of an double linked list which it will act as the database accessed by the server.
* testServerList: It performs a testing of the database implementation in order to ensure that the server uses it in the correct way.
* *client*: It is the one which simulates an user. It has no complexity due to the fact that the only thing that it does is interacting with the following program which it is the one that provides the client side of the services.
* *keys*: As I said in the former paragraph, *keys* is the implementation of the client side of the operations with the server. Moreover, it hides all the complexity of the message passing through queues to the users.

Because of this, the only thing that the user has to do is calling a method, such as *set\_value(…),* and it receives the expected output but he does not know anything about how it was gotten.

* *mensaje*: Mainly it stores the request and response struct in order to been able to use the queues.
* *Makefile*: It is a simple make file which creates the desired executables files as well as the static library *libkeys.a*

Server

It is the program which implements the services offered having the complexity and the real operations of them (not as the client side). Moreover, it handles the message passing through queues in order to been able of responding to the request of the clients.

In order to handle with several clients at the same time and being concurrent, the design approach used is the following one. Once that all the server side queue is created and open an infinite while loop is performed. Inside this loop the first thing that the server does is waiting for a request. At the moment that any request arrived, its client queue is opened in the server side and a new thread is created in order to handle the request.

Moreover, the main execution of the server, using mutual exclusion features, ensures that the request is not overwritten by any other and when this is ensure it performs again the while loop waiting again for a new request.

On the other hand, once that the new thread is created the first thing which it performs is knowing what kind of service is request. In order to perform this checking we compare a special field of the request object that has that information. Furthermore, when we know the service requested and it is performed the last thing to do is just sending back the response to the client.

Finally, in order to send the response the first thing to perform is opening the queue of the client in order to use it as a way of communication. Besides, we check a special field of the response object which has the information about if any error has occurred during the service. The last thing to be done is just sending the response to the client and finishing the thread of the server side.

DoubleLinkedList

This file just stores the implementation of the database of the server as a double linked list. It has been taken copied almost all of the code of the following web page.

<https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_program_in_c.htm>

testServerList

This file just contain a set of test cases for ensuring the correct execution of the database implementation through the server class. Moreover, it has a test for each of the methods asked for implementing in the statement of the exercise.

Furthermore, in order to been able of testing it easily and in a correct way a use the first version of the *server* file called *testServerListAux*. In this implementation the server does not have any mutex nor message queues. The only thing implemented in it are the calls to the database, which is what we want to test.

Client

According to the client code it has no complexity at all. In the client code it is implemented just the simulation of a client’s behavior in order to test the server execution. Because of this, in this file there nothing according to the message passing through queues.

Moreover, in order to communicate the client and the server the following instance is used which is called *keys*.

Keys

Keys code is the interface that the server offers to the users in order to be easier and safer to use it. It provides the interface for all the differences services that the server offers.

Furthermore, it is responsible of creating and managing the queues in order to send the messages as well as closing and unlinking them in order to avoid problems. Besides, it is in charge of checking if it has been any error or problem in the server side in order to inform to the user in that case.

On the other hand, if no error has happened it copies the data obtained from the server response and it send it to the user.

Mensaje

It is just a small file which stores some variables which are not going to be changed (as the name for referring to the server services) and the structures used in the communication through the queues. These stucts are *request* and *response* each struct which its needed fields. Moreover, one interesting field is the *error* field in the response which notifies to the client side that an error has occurred in the server.

Compiling the code

In order to compile the code I have made a very simple make file which compiles all the files and makes the desired executables. These executables are the following ones: *server*, *client*, *testServerList*. Moreover, the static library *libkeys.a* is also created which contains *keys.c* providing the interface to the user for using the server.

Furthermore, in order to create the server executable I have to link the file containing the database implementation called *doubleLinkedList*. Besides, when I create the client object file I have to link the static library previously created, *libkeys.a*, in order to ensure the correct execution and compiling of the client code.

Finally, the executable called *testServerList* just runs a set of test to ensure the correct working of the database implementation. It does not run any test according to message passing due to the fact that it is much harder to write test cases for that kind of problems.

Problems encountered

According to the problems which I found while doing this assignment the main one was regarding to the message queues and how they work. More precisely, the main problem was that I was not able of handling correctly the messages once that I had connect the queues.

In order to solve this problem and to make easier to understand the behavior of the queues and the messages I created two structs. These two struts are one for the request to the server and another for the response of it. Because of this approach it was far easier to handle the messages and the attributes which they have.

Furthermore, another problem which I had according to the messages was the “Message too long “error. In my case, this error was due to the fact that I did not change the message size to the new struct which I created. However, once I discovered why the error happened it was the easiest error to solve.

Regarding to the Makefile, one problem which I had was that I did not really know how to compile several c files in the same command in order to link them. Although the solution was very straightforward once I found the solution it was not so easy to find it. The reason was, in my opinion, that most of the webpages that teaches how to create Makefiles are a bit difficult to understand.