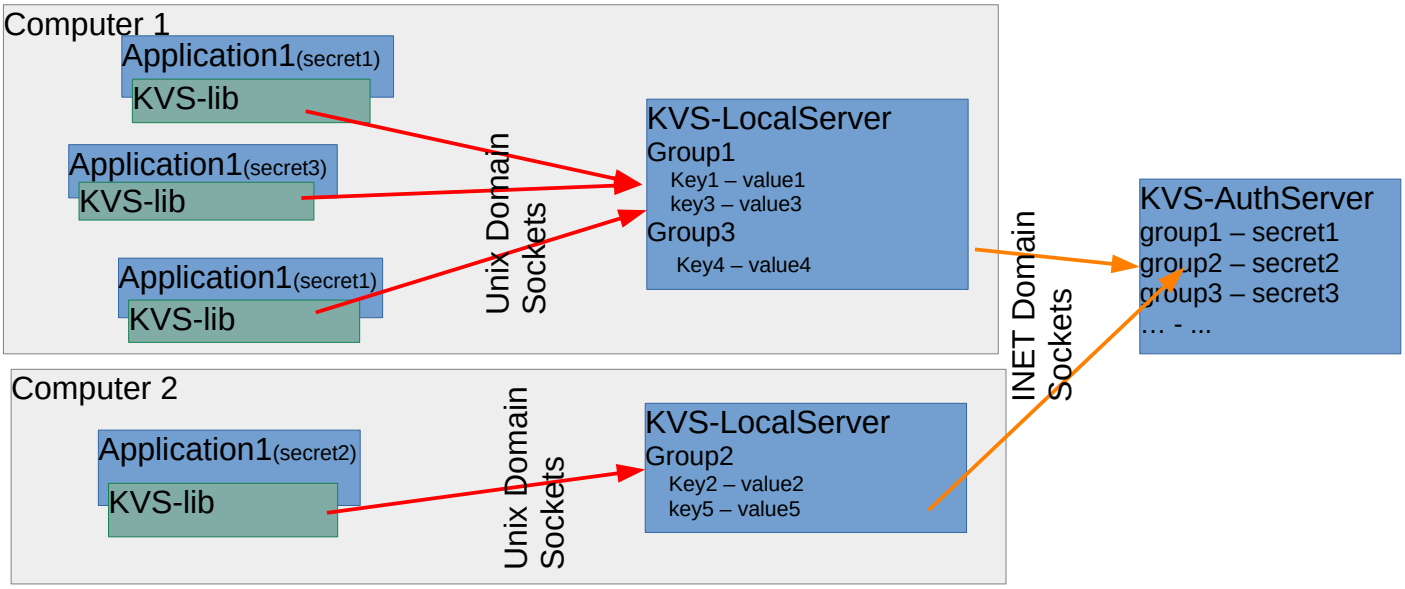
System Programming (MEEC/MEAer)

Project Assignment 2020/2021

In this project the students will implement key value store.

Using this system, application can store values (associated to keys) and share inside a group. Applications belong to a group if they share a secret and use it when connecting to the system.

The components of the KVS (Key-Value System) are as follows:



The applications will use an API implemented by a library to contact and interact with the local server. This server have an user interface for management, receives requests from applications and stores in memory the values. The **KVS-AuthServer** stores information about groups and it secrets.

1 Communication Channels

The applications interact with the library calling C functions declared in the **KVS-lib.h** file and implemented in the **KVS-lib.c** file.

The **KVS-lib** interacts with the **KVS-LocalServer** using UNIX domain stream socket. Before making requests to the **KVS-LocalServer**, each application should call an initialization function from the API that establishes the socket connection and authenticates the application.

The **KVS-LocalServer** interacts with the **KVS-AuthServer** using Internet Domain datagram sockets. The **KVS-AuthServer** can be located in a different computer from the **KVS-LocalServer** and will be accessed by multiple **KVS-LocalServers**.

2 Application interactions

Application use the KVS system to store values associated with key (like an hash-table or dictionary). The data is organized into groups, and only applications belonging to a group can access data of such group.

Groups are created by the administrator (on the **KVS-LocalServer** console) and have a secret associated.

For application to access data, they first need to establishing a connection to the **KVS-LocalServer** providing the groups identifier and the corresponding secret. If they match, then the application can access, manipulate and add data of such group. The application should explicitly close the connection

with the **KVS-LocalServer**.

It is possible for applications to register a callback function that is executed whenever a key-value pair is modified. When the register key-value pair is modified the server **KVS-LocalServer** should send a message to the application and the application runs concurrently with the main the call back function. The call back function is written by the programmer of the application and have the following definition:

- `void f(char * key)`

The registered callback function is invoked by the **KVS-lib** receives as argument the key that was modified and should run in a thread different from the main.

3 KVS-LocalServer

The local server runs on the same computer as applications and allows the sharing of data between applications on the same machine.

This data is stored in a key-value format (like an hash-table or dictionary).

Applications that belong to the same group (establish connection with the correct secret) can access a common set of key-value pairs.

3.1 UI

The **KVS-LocalServer** reads commands from the keyboard in order to configure groups and show status information.

The commands are as follows:

- Create group – receives the identifier of a group (a string) and prints a secret (a string)
- Delete group – removes the group (by deleting the secret and removing all associated data.
- Show group info – prints the following information about a a group:
  - secret
  - number of key-value pairs
- Show application status – lists all currently and past connected application, printing following information:
  - client PID
  - connection establishing time
  - connection close time (if not currently connected).

1.1 Data storage

Each **KVS-LocalServer** stores in memory the key-value pairs. If the server is restarted the information is reset.

1.2 Application authentication

In order to access the data stored in the **KVS-LocalServer**, the application must first connect and provide the group name and the corresponding secrete. If these values match the application can now start to access data associated to such group.

When accepting connection the **KVS-LocalServer** should contact the **KVS-AuthServer** to verify if the provided secret is the one associated with the provided group.

4 KVS-lib

The **KVS-lib** is composed of a set of .c files that should be compiled together with each applications. Students can also generate a **.so** file (as described in week2 laboratory). The corresponding .h file will be provided by the teaching staff and students can not change it.

The .c files should implement the set of functions to be called by applications that will interact with the **KVS-LocalServer** as described next:

- **`int establish_connection (char * group_id, char * secret)`**
  - This function receives as arguments the strings containing the group name and corresponding secret, and tries to open the connection with the **KVS-LocalServer**. If successful, all following operations on key-value pairs are done in then context of the provided group_id.
  - **Return Values** If the secret is correct this function returns 0. If any error occurs the function should return a negative number. For each possible error students should define and return a different return value.
- **`int put_value(char * key, char * value)`**
  - This function tries to assign the provided value to the provided key. If the provided key-pair does not exist in the server, it is created. If the key-value pair already exists in the server it is updated with the provided value.
  - **Return values:** In case of success the function return 1. If any error occurs the function should return a negative number. For each possible error students should define and return a different return value.
- **`int get_value(key, char ** value)`**
  - This function tries to retrieve the value associated to the provided key. If the provided key-pair exists in the server the corresponding value is “returned” through the value argument. This function should do a malloc that will store the value associated with the key.
  - **Return values:** In case of success the function return 1. If any error occurs the function should return a negative number. For each possible error students should define and return a different return value.
- **`int delete_value(key)`**
  - This function tries to retrieve delete the pair key-value associated to the provided key.
  - **Return values:** In case of success the function return 1. If any error occurs the function should return a negative number. For each possible error students should define and return a different return value.
- **`int register_callback(char * key, void (*callback_function)(char *))`**
  - This function tries to register a callback the will later be called. The function receives the key that will be monitored and the pointer to the function that will be executed in the applications. When the value associated with the **key** is changed the function **`callback_function`** should be executed and receives as argument the name of the changed key. Example of a callback function:

```
void f1(char * changed_key){
   printf(“The key with name %s was changed”, changed_key);
```

```
}
```

- **Return values:** In case of success the function return 1. If any error occurs the function should return a negative number. For each possible error students should define and return a different return value.

- int close_connection()
  - This function closes the connection previously opened.
  - **Return values:** In case of success the function return 1. If any error occurs the function should return a negative number. For each possible error students should define and return a different return value.

5 KVS-AuthServer

The **KVS-AuthServer** runs independently of all the processes and can be located in a different computer. Multiple **KVS-LocalServer** can use the **KVS-AuthServer** to store common group/secret information. The manipulation of the data on this server is done in any **KVS-LocalServer** console.

6 Order of development / Functinalities

1. Interaction between KVS-Library and KVS-LocalServer
    1. Definition of messages structure
    2. establish connection (without authentication)
    3. close_connection
2. Interaction between KVS-Library and KVS-LocalServer
    1. Definition of key-value storage on the KVS-LocalServer for one group
    2. put_value
    3. get_value
    4. delete_value
3. Implementation of multiple groups
    1. establish connection (with group name but without authentication)
    2. Definition of key-value storage on the KVS-LocalServer for multiple groups
    3. put_value
    4. get_value
    5. delete_value
4. Implementation of the KVS-LocalServer UI
    1. Show application status
        1. Define data structures to store necessary information
        2. change establish_connection and close_connection
5. Implementation of KVS-Aauth
    1. Definition of the internal data structures of KVS-Auth
    2. Definition of communication protocol between KVS-LocalServer and KVS-AuthServer
    3. establish_connection (with authentication)
    4. Implementation of the KVS-LocalServer UI
        1. Create group
        2. Delete Group
        3. Show group info
6. Implementation of the callback
    1. register_callback on the KVS_library
        1. Definition of data structures and components on the KVS-Libarry
        2. Definition of communication protocol between KVS-Libary and KVS-LocalServer
    2. Data structures and components on the KVS-LocalServer
7. Synchronization

8. Error handling

7 Multi-threading/ychnronization

Since all servers (LocalServe and AuthServer) should receive requests from multiple clients it is necessary to guarantee that any race condition is solved.

In the case of the KVS-LocalServer it is necessary to use synchronization mechanisms (for instance mutexes) to implement critical regions.

Depending on the way the KVS-authServer is implemented, it may be necessary to also use mutexes to define critical regions.

8 Error validation

It is fundamental that most possible errors are gracefully handled.

The the KVS-Library function should return various error codes, but the internal code of the Library and Servers should do extra error validations.

2 Code organization

The code of the various components (KVS-Library, KVS-LocalServer and KVS-AuthServer) should be split into various functions. These functions should be grouped into internal modules that will have a specific functionality/responsibility in the project.

Some of the modules can be implemented as threads.

9 Evaluation and dates

Students should finalize and submit the project and report during the week on May the 4th. The presentation of the projects will be done after the delivery project of the in a date to arrange with the teaching staff.

9.1 Project submission

The project will be submitted on FENIX. Students should provide a report and an archive of all the code. At the same time students should select one date for the evaluation (link will be provided at the FENIX).

9.2 Report

Students should present a reports explaining the developed system. The report should contain clearly the following information:

- Graphical representation of the architecture and components of the system. Students should expand the figure presented in this report stating what functionality are present on each processes/components/modules.
- The way thread management is perform should be described. For each process student should list all thread, describe their functionality and explaining how they are created.
- The interaction protocols between the components (KVS-library, KVS-LocalServer and KVS-AuthServer) and their internal modules should be represented using Sequence Diagrams.
- The internal data structures
- How the various components are implemented with respect to functions, threads and data structures.

- How remote communication is performed
- What data is exchanged between components
- How synchronization is implemented.

In the final section of the report students should clearly describe in a form what functionalities were implemented (form to be provided).

10 Project presentation

For the presentation of the project students should prepare a set of clients and usage steps that allow the teaching staff to evaluate the various functionality.

11 Evaluation criteria

The evaluation criteria for project grading will be published in the near future, but will be related to the functionalities implemented (as described in Section 6) and the quality of the developed system.