C example

Table of contents

1. Library lib_KalimaMQC

This directory contains all the headers of the library, as well as a static archive lib_KalimaMQC.a. This archive will be used to create the executable of the project (MainProject). The headers allow us to call the methods that will be useful in our example.

2. Principal project

a. The configuration file

```
LedgerName=KalimaLedger
NODE_NAME=Node Example
NotariesList=167.86.124.188:9090 62.171.130.233:9090 62.171.131.157:9090 144.91.108.243:9090
FILES_PATH=log/
SerialID=LouisVM
```

The meanings of the different configurations in this config file are :

- LedgerName ➔ This is the name of the Ledger to which we will connect. For our example this is not useful.
- NODE_NAME ➔ This is the name of the node that we create. This is not useful for our exemple.
- NotariesList ➔ This is the list of nodes you have to connect to, in order to communicate with the tuto Kalima blockchain. If you want to connect to another Kalima blockchain, you just have to change the notaries in this file. Between each node, you have to put a space.
- FILES_PATH ➔ This is the directory in which we find the log files and the file containing the encrypted deviceID. In our example, the log folder will be created in the MainProject directory at running of the executable.
- SerialID ➔ This SerialID will be used to authenticate to the Kalima blockchain. It must correspond to the name of a device on the blockchain.

b. The main.mk makefile

```
CC=gcc -pthread
CFLAGS=-W -Wall
EXEC= Exec

all: $(EXEC)

NodesChatC.o: main.c
    $(CC) -o main.o -c main.c $(CFLAGS)

Exec: main.o
    $(CC) -o main.run main.o -L. ../lib_KalimaMQC/lib_KalimaMQC.a $(CFLAGS)

clean:
    rm -rf *.o
    rm -r log/

mrproper: clean
    rm -rf $(EXEC)
```

The makefile allows you to create an object for your main. Then, from this object and the library archive, you can create your executable "main.run".

c. The main of the project

```
Node *node;
if(argc>1){
    node = create_Node(argv[1]);
    printf("%s Loaded\n", argv[1]);
}
else{
    node = create_Node(pre_config);
    printf("%s Loaded\n", pre_config);
}
printf("GO!\n");
Menu();
```

The node will be created here. By default, the node will use the file "config.txt" as the config file. However, if you enter a config file as a parameter to run the program, it will be used instead. The node will be created with the parameters of the config file.

When creating the node, a random deviceID will be created which will be encrypted and written to the file "DeviceID. This file will be created in the "FILES_PATH" folder (by default "log/"). If this encrypted file already exists, it will not be recreated. The node will just decrypt the file and use the recovered deviceID. This deviceID, in conjunction with the SerialID, will allow the blockchain to identify your node and allow you to write to it.

Finally, it will connect directly to the nodes configured in the config file.

After creating the node, the example offers two choices to the user :

- Default message

```
char str[100];
char temp;
printf("Message to send to Blockchain :\n");
scanf("%c", &temp);                              //Clear buffer
fgets(str, sizeof(str), stdin);                  //Get input
send_Message_to_Notaries(node, str);             //Send message
printf("Message sent\n");
sleep(2);                                        //Waiting for response (in logs)
```

This choice proposes to enter a message in the terminal. This message will then be sent to the cachepath « /sensors » with the key « key ». The « sleep(2) » at the end is used to wait for 2 seconds to receive the response from the blockchain. You can see the reception of the response in the logs.

- Configurable message

```
proplist kProps = Initiate_prop_List();
kProps = set_prop(kProps, "ttl", 3, "-1", 2); // will give ttl=-1, meaning the message will stay in blockchain (ttl = time to live)
char type = encode_Type(PUB); //Publish
char cachePath[100], key[100], msg[100];
char temp;
printf("Type the cachePath you want to interact with :\n");
scanf("%c", &temp);                                 //Clear buffer
fgets(cachePath, sizeof(cachePath), stdin);
strtok(cachePath, "\n");
printf("Type the key of you choice :\n");
fgets(key, sizeof(key), stdin);
strtok(key, "\n");
printf("Type the message of you choice :\n");
fgets(msg, sizeof(msg), stdin);
strtok(msg, "\n");
KMsg *kmsg = getMessage(node->devid, UUID_SIZE, type, cachePath, strlen(cachePath), key, strlen(key), 1, msg, strlen(msg), kProps);
send_to_Notary_Nodes(node, kmsg->Kmessage);  //Send message
printf("Message sent\n");
sleep(2);                                   //Waiting for response (in logs)
```

Here you will build the message yourself from scratch.

The kProps created here are the parameters to be transmitted to the blockchain for the message. Here for example, we set the "time to live" of our message. When it is set to "-1" like here, we signal to the blockchain that the message must remain without disappearing. However, if we set it to "5", it will remain for 5 seconds before being deleted automatically.

Then we get the cache path, the key and the message from the user's terminal.

After that, we create a KMessage with our parameters and we send this KMessage to the notary nodes.

The "sleep" is still there to wait for the response.

3. Program execution

To execute the example, you just have to open a terminal and to access to the « MainProject » folder.

Then type « make -f main.mk » to compile the makefile seen before. This command will create the executable file « main.run » which will allow you to launch the program (as well as a file main.o which can be deleted).

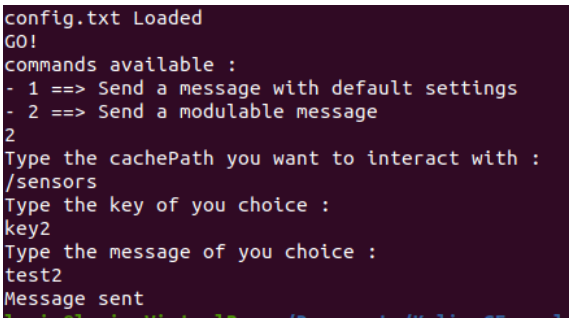
Finally, you must type "./main.run" to launch the program.

```
config.txt Loaded
GO!
commands available :
- 1 ==> Send a message with default settings
- 2 ==> Send a modulable message
```

Once the program is launched, this is the result that should appear. Here we see that the file « config.txt » is loaded for the creation of the node. At this step, the node is created and connected to the blockchain. By typing « 1 » or « 2 » you can execute one of the two options seen above.

```
config.txt Loaded
GO!
commands available :
- 1 ==> Send a message with default settings
- 2 ==> Send a modulable message
1
Message to send to Blockchain :
test
Message sent
```

Here, we have an example of the default message. The message « test » is sent in the cachepath « /sensors » with the key « key ».



Here, we have an example of a configurable message. The message « test2 » is sent in the cachepath « /sensors » with the key « key2 ».

4. Possible errors

In case that your message does not appear in the blockchain, here are some possible errors :

- Check that the SerialID in the config file is the same as the one entered in the blockchain.
- If you delete the DeviceID file, the new one calculated will be different. You will have to recreate the SerialID because the associated DeviceID will not be the same.
- Check that the name of the entered cache path is the right one.