Exemple C

Table des matières

1. Librairie (libs/KalimaLib)

Cette librairie contient tous les headers de la librairie, ainsi que deux archives statique lib_KalimaNodeLib.a et lib_KalimaNodeLib32.a. Ces archives nous

serviront lors de la création de l'exécutable dans le projet (src). Les headers nous serviront à appeler les méthodes qui nous seront utiles dans notre exemple.

2. Fichier config (etc/cfg)

```
LedgerName=KalimaLedger
NODE_NAME=Node Example
NotariesList=167.86.103.31:8080 5.189.168.49:8080 173.212.229.88:8080 62.171.153.36:8080 167.86.124.188:8080
FILES_PATH=log/
PRIVATE_KEY_FILE=RSA/private.pem
PUBLIC_KEY_FILE=RSA/public.pem
BLOCKCHAIN_PUBLIC_KEY_FILE=RSA/public.pem
SerialID=louisTest
```

Voyons l'utilité des différentes configurations :

- LedgerName -> C'est le nom du Ledger auquel nous allons nous connecter. Pour l'exemple ce n'est pas très utile.
- NODE_NAME -> C'est le nom du node que nous créons. Ce n'est pas non plus très utile pour notre exemple
- NotariesList -> C'est la liste des nodes auxquels nous devons nous connecter pour communiquer avec la blockchain tuto. Si vous souhaitez vous connecter à une autre blockchain, il suffit de changer les notaries ici. Entre chaque node, il faut mettre un espace.
- FILES_PATH -> C'est le directory dans lequel on pourra trouver les fichers de logs. Dans notre exemple, le dossier log se créera au lancement dans le directory
- KEY_FILES -> Ce sont les chemins des différents fichiers de cryptage RSA. Ces fichiers sont nécessaires pour communiquer avec la Blockchain. Ils seront également créés automatiquement au lancement du projet.
- SerialID -> Ce SerialID nous servira d'identification auprès de la blockchain. Il doit être autorisé par la blockchain. Ce sera surement la seule ligne que vous devrez modifier pour mettre l'ID que vous souhaitez., merci de contacter un de nos administrateurs (jerome.delaire@kalima.io, tristan.souillard@kalima.io)

3. Projet principal (src)

a. Makefile

```
CC=gcc -pthread
CFLAGS=
LDFLAGS=-lcrypto -lm
EXEC= Exec

LIBRARY=lib_KalimaNodeLib.a

all: $(EXEC)

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

Exec: main.o
    cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi; mkdir objects; cd objects; ar -x ../$(LIBRARY)
    $(CC) -o main.run main.o ../libs/KalimaLib/objects/*.o $(LDFLAGS)
    cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi

clean:
    rm -rf *.o
    rm -rf main.run
    if [ -d log ]; then rm -rf log; fi
    cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi

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

Le makefile nous permet de créer un objet pour notre main. Ensuite, à partir de cet objet et de l'archive de la librairie, nous pouvons créer notre exécutable « main.run ».

Ici, la librairie utilisée par défaut est « lib_KalimaNodeLib.a » qui a été compilée sur une architecture linux 64bits. Il est également possible de remplacer par « lib_KalimaNodeLib32.a » pour créer l'exécutable sur une architecture 32bits.

b. Main du projet

```
Node *node = create_Node("../etc/cfg/config.txt", NULL);
printf("Config loaded\n");
Connect_to_Notaries(node, NULL);
sleep(2);
printf("GO!\n");
```

Ici, nous créons notre node. Par défaut, le node utilisera le fichier « config.txt » comme fichier de config. Ici, il est placé dans le dossier etc/cfg (.. sert à revenir dans le dossier précédent).

Lors de la création du node, nous allons également créer un deviceID aléatoire qui sera crypté et écrit dans le fichier « DeviceID » qui sera créé dans le dossier src. Si ce fichier crypté existe déjà, il ne sera pas recréé. Le node va juste décrypter le fichier et utiliser le deviceID récupéré. Ce deviceID, en lien avec le SerialID, permettra à la blockchain d'identifier notre node et de nous permettre d'écrire dans celle-ci. Un dossier RSA sera également créé contenant une clé publique et une clé privée servant au cryptage des communications avec la blockchain.

Les paramètres « NULL » que l'on peut voir ici correspondent au fait que l'on peut permettre au node d'utiliser des scipts LuaJIT avec un callback. Vu que dans cet exemple nous n'en utilisons pas, il faut mettre NULL.

Enfin, il se connectera directement aux nodes écrits dans le fichier config.

Après la création du node, l'exemple propose deux choix à l'utilisateur :

- Envoi de 10 messages par défaut

```c
void send_10_messages(Node *node){
    for(int i=0 ; i<10 ; i++) {
        uint8_t body_size = get_int_len(95+i);
        char body[body_size];
        snprintf(body, body_size+1, "%d", 95+i);
        char key[13];
        snprintf(key, 13, "%s%d", "temperature", i);
        put_msg_with_ttl(node->clone, "/sensors", 8, key, 12, body, body_size, 10);
        printf("Sending value %s to key %s on address : /sensors\n", body, key);
        sleep(1);
    }
}
```

Ce choix envoie 10 messages prédéfinis à la blockchain sur l'adresse « /sensors ». Les 10 messages auront une clé allant de temperature0 à temperature9 et une valeur allant de 95 à 104. Chaque message restera 10 secondes dans la blockchain avant de se supprimer automatiquement

- Message entièrement configurable

```c
void send_modulable_message(Node *node){
    char temp;
    char choice[10] = {}, address[100] ={}, key[100] ={};
    scanf("%c", &temp);                                 //Clear buffer
    while(strncmp(choice, "a", 1) != 0 && strncmp(choice, "d", 1) != 0){
       printf("Do you want to add (a) or delete (d) ?\n");
       fgets(choice, sizeof(choice), stdin);

    }
    printf("Type the address you want to interact with :\n");
    fgets(address, sizeof(address), stdin);
    strtok(address, "\n");
    printf("Type the key of you choice :\n");
    fgets(key, sizeof(key), stdin);
    strtok(key, "\n");

    if(strncmp(choice, "a", 1) == 0){
        char msg[100];
        printf("Type the message of you choice :\n");
        fgets(msg, sizeof(msg), stdin);
        strtok(msg, "\n");
        put_msg_default(node->clone, address, strlen(address), key, strlen(key), msg, strlen(msg));
        printf("Sending value %s to key %s on address : %s\n", msg, key, address);
    }
    if(strncmp(choice, "d", 1) == 0){
        remove_msg(node->clone, address, strlen(address), key, strlen(key));
        printf("Deleting key %s on address : %s\n", key, address);
    }
}
```

Ici nous construisons nous même le message de 0.

On propose tout d'abord à l'utilisateur de choisir entre ajouter ou supprimer un message. Ensuite on récupère l'adresse, la clé et le message depuis le terminal

de l'utilisateur. Enfin on envoie le message. Contrairement à l'exemple précédent, le message restera ici indéfiniment.

4. Exécution du programme

Pour exécuter l'exemple, il suffit d'ouvrir un terminal et de se placer dans le dossier « src ».

Il suffit ensuite simplement de taper « make» pour lancer le makefile vu précédemment. Cette commande créera le fichier exécutable « main.run » qui nous permettra de lancer le programme (ainsi qu'un fichier main.o qui peut être supprimé, le dossier DevID et le dossier RSA).

Vous pouvez également lancer avec la librairie 32bits en tapant « make LIBRARY= « lib_KalimaNodeLib32.a » » si vous n'avez pas déjà modifié le makefile.

```
e/src$ make
gcc -pthread -o main.o -c main.c
cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi; mkdir objects;
 cd objects; ar -x ../lib_KalimaNodeLib.a
gcc -pthread -o main.run main.o ../libs/KalimaLib/objects/*.o -lcrypto -lm
cd ../libs/KalimaLib; if [ -d objects ]; then rm -rf objects; fi
```

Enfin, il faut écrire « ./main.run » pour lancer le programme.

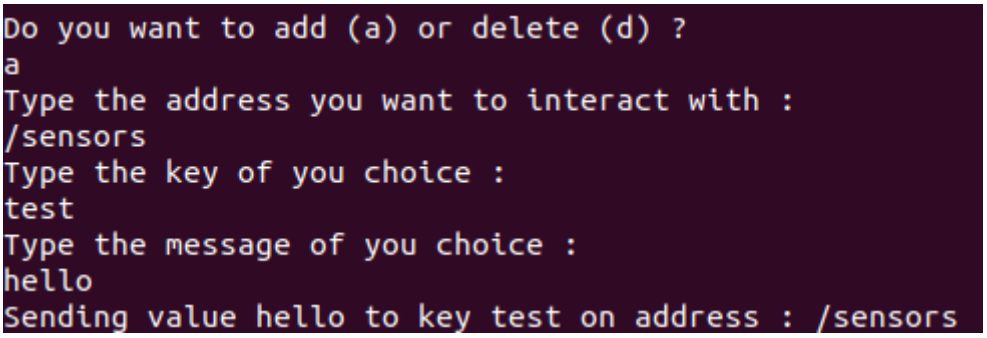
```
e/src$ ./main.run
Config loaded
GO!
commands available :
- 1 ==> Send 10 messages
- 2 ==> Send a modulable message
- 3 ==> Close
```

Une fois le programme lancé, voici ce qui devrait apparaître. Nous voyons ici que le fichier de config est chargé pour la création du node. À ce niveau-là, le node est créé et connecté à la blockchain. En tapant « 1 », « 2 » ou « 3 » on peut ainsi exécuter l'une des options vues ci-dessus.

Voici ce que nous obtenons avec la première option :

```
Sending value 95 to key temperature0 on address : /sensors
Sending value 96 to key temperature1 on address : /sensors
Sending value 97 to key temperature2 on address : /sensors
Sending value 98 to key temperature3 on address : /sensors
Sending value 99 to key temperature4 on address : /sensors
Sending value 100 to key temperature5 on address : /sensors
Sending value 101 to key temperature6 on address : /sensors
Sending value 102 to key temperature7 on address : /sensors
Sending value 103 to key temperature8 on address : /sensors
Sending value 104 to key temperature9 on address : /sensors
```

Et voici ce que nous avons avec la deuxième option :



Ici nous avons décidé d'envoyer la valeur « hello » pour la clé « test » sur l'adresse « /sensors »

La troisième option sert juste à fermer le programme.

5. Problèmes possibles

Si votre message n'apparaît pas dans la blockchain, voici quelques possibles erreurs :

- Vérifiez que le SerialID dans le fichier config est le même que celui entré dans la blockchain.
- Si vous supprimez un fichier DeviceID ou RSA, le nouveau calculé sera différent. Il faudra donc refaire le SerialID car le DeviceID associé ne sera plus le même.
- Si vous avez fait des modifications sur le main qui ne sont pas prises en compte, faîtes un « make clean » avant de refaire « make » pour que toutes les modifications soient bien prises en compte.
- Si vous avez d'autres problèmes, vous pouvez regarder dans les logs quand le programme a des problèmes et nous contacter.