ELEC 477

Kevin Yu: 20203451

Raatik Sharma: 20120770

Ainsley Taylor: 20210012

2024-02-05

Test 1: Testing Integrated DNS System

"For testing the integrated system, you should test two different kv servers running as different service names, and that theclients correctly find them by service name."

The test results for Test 1 are shown in the "Test1.txt" file in the repository. In this file it showcases the system logs of the DNS program. In main.cpp file the test case illustrates a DNS server being configured with the following setttings:

```
// KY: ------------------------------------ DNS SERVER ------------------------------------ //
shared_ptr<DirSvcServer> dnsServer = make_shared<DirSvcServer>("dns_server");

dnsServer->setAddress("10.0.0.69");
dnsServer->setPort(8081);
dnsServer->setSvcName("DNS");
dnsServer->init(); // Initializes the simulated DNS config and other activities??
dnsServer->startServices(); // Starts up the list of services by creating a new thread for each service so aka one service for each server.
```

Additionally, two servers being instantiated with different service names, server names, Ip addresses and port numbers:

```
// KY: ------------------------------------ KVSERVER1 ------------------------------------ //
// KY: Every instance of a server that is created, a new service is added to the node's "services" list
shared_ptr<KVServer> kvServer = make_shared<KVServer>("kvserver");

kvServer->setAddress("10.0.0.12");
kvServer->setPort(8080);
kvServer->setSvcName("email");
kvServer->setDBMFileName("server");
kvServer->init();          // Initializes the simulated DNS config and other activities??
kvServer->startServices(); // Starts up the list of services by creating a new thread for each service so aka one service for each server.

// KY: ------------------------------------ KVSERVER2 ------------------------------------ //
// KY: Every instance of a server that is created, a new service is added to the node's "services" list
shared_ptr<KVServer> kvServer2 = make_shared<KVServer>("kvserver2");

kvServer2->setAddress("10.0.0.10");
kvServer2->setPort(8089);
kvServer2->setSvcName("taxi");
kvServer2->setDBMFileName("server2");
kvServer2->init();          // Initializes the simulated DNS config and other activities??
kvServer2->startServices(); // Starts up the list of services by creating a new thread for each service so aka one service for each server.
```

Lastly, two clients were instantiated to talk to one of these two servers by utilizing the directory service name stub which would talk to the DNS server and DNS stub:

```
// KY: ------------------------------------ INITIALIZING CLIENT 1 ------------------------------------ //
std::cout << "\nMain: **********" << std::endl;
std::cout << "Main: init client\n" << std::endl;
std::this_thread::sleep_for(std::chrono::milliseconds(1000));
shared_ptr<KVClient1> kvClient = make_shared<KVClient1>("kvclient");
kvClient->setAddress("10.0.0.14");
kvClient->setSvcName("email");
kvClient->init();

std::cout << "Main: **********" << std::endl;
std::cout << "Main: starting client" << std::endl;
vector<shared_ptr<thread>> clientThreads;
{
    // need a scope for the lock guard.
    // if this doesn't work put it in a function
    std::lock_guard<std::mutex> guard(nodes_mutex);

    shared_ptr<thread> t = make_shared<thread>([kvClient]()
                                               { kvClient->execute(); });

    clientThreads.push_back(t);
    nodes.insert(make_pair(t->get_id(), kvClient));
    names.insert(make_pair(t->get_id(), "kvclient"));
}
```

```
// // KY: ------------------------------------ INITIALIZING CLIENT 2 ------------------------------------ //
std::cout << "Main: **********" << std::endl;
std::cout << "Main: init client" << std::endl;
std::this_thread::sleep_for(std::chrono::milliseconds(1000));
shared_ptr<KVClient1> kvClient2 = make_shared<KVClient1>("kvclient2");
kvClient2->setAddress("10.0.0.11");
kvClient2->setSvcName("taxi");
kvClient2->init();

std::cout << "Main: **********" << std::endl;
std::cout << "Main: starting client 2" << std::endl;
vector<shared_ptr<thread>> clientThreads2;
{
    // need a scope for the lock guard.
    // if this doesn't work put it in a function
    std::lock_guard<std::mutex> guard(nodes_mutex);

    shared_ptr<thread> t = make_shared<thread>([kvClient2]()
                                               { kvClient2->execute(); });

    clientThreads2.push_back(t);
    nodes.insert(make_pair(t->get_id(), kvClient2));
    names.insert(make_pair(t->get_id(), "kvclient2"));
}
```

From the system logs in "Test.txt" we can see that the servers registered their services with the DNS server:

```
21  ---------- DNS SERVER RECEIVED SOMETHING ----------
22
23  DIR_SVC_SERVICE: register message requested
24
25  DIR_SVC_SERVICE: THIS IS THE SERVICE KEY----------->: taxi
26  DIR_SVC_SERVICE: THIS IS THE SERVER KEY----------->: kvserver2
27  DIR_SVC_SERVICE: THIS IS THE SERVER PORT----------->: 8089
28
29
30  DIR_SVC_SERVICE: END OF REGISTER SERVICE
31
32  waiting for call from client
33
34  waiting for call from client
35  Main: ************************************
36  Main: starting client
37  Main: ************************************
38  Main: init client
39  //------------------------------------ KV CLIENT TEST ------------------------------------//
40
41  ---------- DNS SERVER RECEIVED SOMETHING ----------
42
43  DIR_SVC_SERVICE: register message requested
44
45  DIR_SVC_SERVICE: THIS IS THE SERVICE KEY----------->: email
46  DIR_SVC_SERVICE: THIS IS THE SERVER KEY----------->: kvserver
47  DIR_SVC_SERVICE: THIS IS THE SERVER PORT----------->: 8080
48
49
50  DIR_SVC_SERVICE: END OF REGISTER SERVICE
51
52  waiting for call from client
53
```

After the servers registered their services, the clients can then use the DNS service to lookup the server address information based on the corresponding choosen service name:

```
54  ---------- DNS SERVER RECEIVED SOMETHING ----------
55
56  DIR_SVC_SERVICE: Search message requested
57  DIR_SVC_SERVICE: Searching for following service -------> email
58
59  DIR_SVC_SERVICE: Search Resolved Status -------> 1
60  DIR_SVC_SERVICE: Search Resolved Server Name -------> kvserver
61  DIR_SVC_SERVICE: Search Resolved Server Port-------> 8080
62
63  DIR_SVC_SERVICE: END OF SEARCH SERVICE
64
65  waiting for call from client
66
67  KV_CLIENT_STUB: THE SERVER NAME SET IN KV CLIENT STUB IS --------> kvserver
68  KV_CLIENT_STUB: THE SERVER PORT SET IN KV CLIENT STUB IS --------> 8080
69
70  waiting for call from client
71  Main: ************************************
72  Main: starting client 2
73  Main: ************************************
74  Main: waiting for clients to finish
75  ---------- DNS SERVER RECEIVED SOMETHING ----------
76
77  DIR_SVC_SERVICE: Search message requested
78  DIR_SVC_SERVICE: Searching for following service -------> taxi
79
80  DIR_SVC_SERVICE: Search Resolved Status -------> 1
81  DIR_SVC_SERVICE: Search Resolved Server Name -------> kvserver2
82  DIR_SVC_SERVICE: Search Resolved Server Port-------> 8089
83
84  DIR_SVC_SERVICE: END OF SEARCH SERVICE
85
86  waiting for call from client
87
88  KV_CLIENT_STUB: THE SERVER NAME SET IN KV CLIENT STUB IS --------> kvserver2
89  KV_CLIENT_STUB: THE SERVER PORT SET IN KV CLIENT STUB IS --------> 8089
90
91  put message requested
92  put message requested
93  waiting for call from client
```

Finally, we can see the KVRPC function properly as each client is able to put and get corresponding values from each server:

```
put message requested
put message requested
waiting for call from client
//------------------------------------ END OF PUT --------------------------------------//

MESSAGE STORED IN DB------> This is a test! Kevin!
STATUS OF PUT IS: 1


//------------------------------------ END OF PUT --------------------------------------//

waiting for call from client
MESSAGE STORED IN DB------> LOOOOOOOOOOOOOOOOOOOOOL
get message requested
get message requested
waiting for call from client
waiting for call from client
//------------------------------------ END OF GET --------------------------------------//

STATUS OF GET IS: 1
STATUS OF GET IS: This is a test! Kevin!


DIR_SVC_CLIENT_STUB: Key sent to delete ---------> email
//------------------------------------ END OF GET --------------------------------------//


DIR_SVC_CLIENT_STUB: BYTES SENT ----------> ---------------- DNS SERVER RECEIVED SOMETHING ----------------

19DIR_SVC_SERVICE: delete message requested

DIR_SVC_SERVICE: deleting following service ------->email
DIR_SVC_SERVICE: Deleted the following service -------->email

```

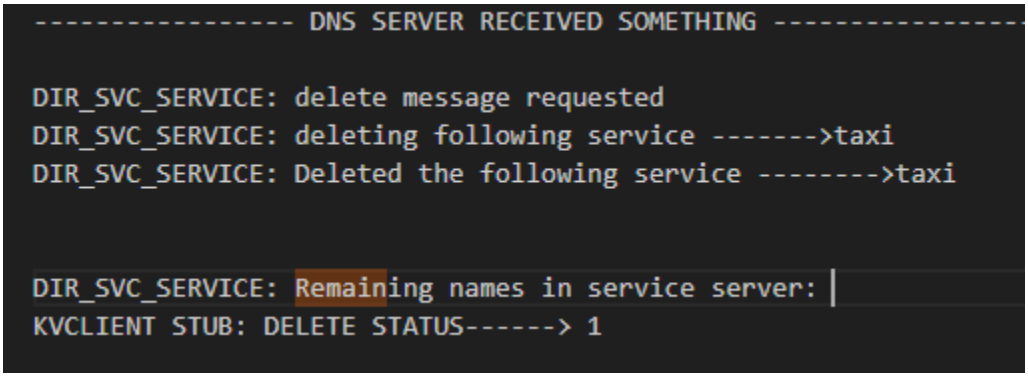
Test 2: Deleting Service from Service Server

"You should check that when a kv server exits, that the name of the service is no longer in the service server."

Using the same test results shown in "Test.txt" and the same main.cpp class we can see what happens when a service is shutdown and deleted from the service server. In the system logs every time we shutdown/delete a service we iterate through the remaining services left in the server:

```
118   ----------------- DNS SERVER RECEIVED SOMETHING -----------------
119
120   DIR_SVC_SERVICE: delete message requested
121
122   DIR_SVC_SERVICE: deleting following service ------->email
123   DIR_SVC_SERVICE: Deleted the following service -------->email
124
125
126   DIR_SVC_SERVICE: Remaining names in service server:
127   Key: taxi, Server Name: kvserver2, Server Port: 8089
128
129
```

Line 127 showcases the remaining services stored on the service server. Once the last service is deleted the following is shown:



We can see that our system logs show that there are no more services registered in the DNS server. Therefore when a kv server exits, the name of the service is no longer in the service server.