ELEC 477

Kevin Yu: 20203451

Raatik Sharma: 20120770

Ainsley Taylor: 20210012

2024-02-05

For example: "File test1.txt
contains the output of our first test. It shows that the value 'Smith Engineering is great" was stored with the key 1, and successfully retrieved using the key 1". You will need to have appropriate print statement in the client so that the output shows that the test is successful.

Test 1

1. *The first test should have a simple client that puts a value and retrieves it and checks that it got the same value back.*

Test 1 asks us to perform a PUT and GET RPC. In our client.cpp file we called a PUT request with a value of “kevin/0yu” with a key of 7. Test1.txt shows the output of our results when we do a GET request as shown below. This illustrates that the RPC methods for PUT and GET work correctly since we fetch the correct value that was associated with the key 7.

```
--------------- E1SERVICE: Get Request Finished ---------------
Client's GET status: 1
Client's GET value: kevinyu
Client's GET length: 8
Main: ************************************
Main: calling stop services on server
Main: ************************************
Main: waiting for threads to complete
Main: shutting down protobuf library
```

Test 2

2. You must test that data containing null bytes is stored and retrieved correctly.

Test 2 asks us to see if our RPC works with storing and returning null bytes. In this test we do the same PUT and GET request as in Test 1. But we illustrate that storing and retrieving null bytes works correctly since “kevin\0yu” contains a null byte in the middle indicating that the end of the string is after the name “kevin” and “yu” would not be stored theoretically. However, we see from the image below that our GET request does indeed return back “kevinyu” meaning our RPC method can store and retrieve null bytes.

```
// Setup network configuration using command line args
int res = network_init(argc, argv);
std::stringstream ss;

// Start up the server
std::cout << "Main: **********" << std::endl;
std::cout << "Main: starting server" << std::endl;

// Initializing the server with its own address
shared_ptr<E1Server> e1Server = make_shared<E1Server>("e1server");
e1Server->setAddress("10.0.0.2");
e1Server->setGdbmFile("dean.db");
e1Server->startServices();

// Wait for the servers to setup before initalizing the client
std::this_thread::sleep_for(std::chrono::milliseconds(50));

// Initialize the client
std::cout << "Main: **********" << std::endl;
std::cout << "Main: init client" << std::endl;
shared_ptr<E1Client> e1Client = make_shared<E1Client>("e1client");
e1Client->setAddress("10.0.0.3");
e1Client->setServerAddress("10.0.0.2");

// Starting Client to send messages to the server
std::cout << "Main: **********" << std::endl;
std::cout << "Main: starting client" << std::endl;
shared_ptr<thread> t;

// Create a guard scope to add client to global variables and call its start method
{
  // 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);

  // Spin off a new thread and call the start() method on the client
  t = make_shared<thread>([e1Client]()
                          {
        try{
          e1Client -> start();
        } catch (exitThread & e){ } });

  // Critical section
  nodes.insert(make_pair(t->get_id(), e1Client));
  names.insert(make_pair(t->get_id(), "e1client"));
}

// Wait for the threads to finish its execution before proceeding with the main thread
std::cout << "Main: **********" << std::endl;
std::cout << "Main: waiting for clients to finish" << std::endl;
t->join();

// Stops all services on the servers by setting alive flag to false.
std::cout << "Main: **********" << std::endl;
std::cout << "Main: calling stop services on server" << std::endl;
e1Server->stopServices();

// Wait for all server threads to complete
std::cout << "Main: **********" << std::endl;
std::cout << "Main: waiting for threads to complete" << std::endl;
e1Server->waitForServices();

google::protobuf::ShutdownProtobufLibrary();
std::cout << "Main: shutting down protobuf library" << std::endl;

return 0;
```

```
#include "e1client.hpp"
#include "data.pb.h"
using namespace std;
using namespace std::string_literals;

#define close mclose

/**
 * Sets the server address attribute so the client stub can create a socket for the server
 */
void E1Client::setServerAddress(char *serverAddress)
{
  this->serverAddress = serverAddress;
}

/**
 * Defines the RPC calls that the client will invoke
 */
void E1Client::start()
{
  // Create an instance of the client stub
  clientStub = make_shared<ClientStub>();
  clientStub->setServerAddress(serverAddress);
  // Call the put method from the client stub
  Data::put_response dataPut = clientStub->put(13, reinterpret_cast<const uint8_t *>("kevin\0yu"), 10);
  std::cout << "E1CLIENT's PUT status: " << dataPut.success() << std::endl;

  // Call the get method from the client stub
  Data::get_response dataGet = clientStub->get(13);

  // Print out the results form the get request
  std::cout << "Client's GET status: " << dataGet.status() << std::endl;
  std::cout << "Client's GET value: " << dataGet.value() << std::endl;
  std::cout << "Client's GET length: " << dataGet.value_length() << std::endl;
}
```

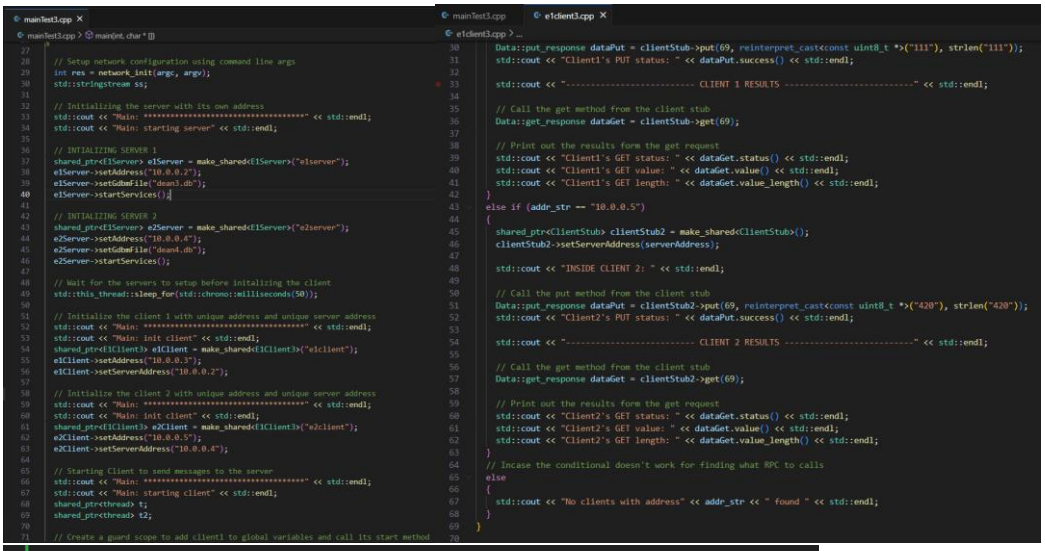
```
-------------- E1SERVICE: Get Request Finished --------------
Client's GET status: 1
Client's GET value: kevinyu
Client's GET length: 8
Main: **********
Main: calling stop services on server
Main: **********
Main: waiting for threads to complete
Main: shutting down protobuf library
```

Test 3

3. You should have tests with two servers and two clients to show independence.

In Test 3, we setup two independent clients and two independent servers each with their unique addresses and each server has its own unique database. We call a PUT request for each client with the exact same key “69” but client 1 has a value of “111” and client 2 has a value of “420”. When we call the

GET request on each client with the same key of “69” we see that each client returns their respective value that they stored with their PUT request. This illustrates that each client and server pair are independent of one another.



```
---------------------------- CLIENT 2 RESULTS ----------------------------

--------------- E1SERVICE: Put Request Finished ---------------
CLIENT STUB: Success status of the put request:0
Client1's PUT status: 0
---------------------------- CLIENT 1 RESULTS ----------------------------
E1SERVICE: server received 12 bytes
E1SERVICE: Client Address 10.0.0.5
E1SERVICE: server received 12 bytes
E1SERVICE: Client Address 10.0.0.3
--------------- E1SERVICE: Get Request Finished ---------------
Client2's GET status: 1
Client2's GET value: 420
Client2's GET length: 8
--------------- E1SERVICE: Get Request Finished ---------------
Client1's GET status: 1
Client1's GET value: 111
Client1's GET length: 8
Main: ************************************
Main: calling stop services on server
Main: ************************************
Main: waiting for threads to complete
Main: shutting down protobuf librar
```

Test 4

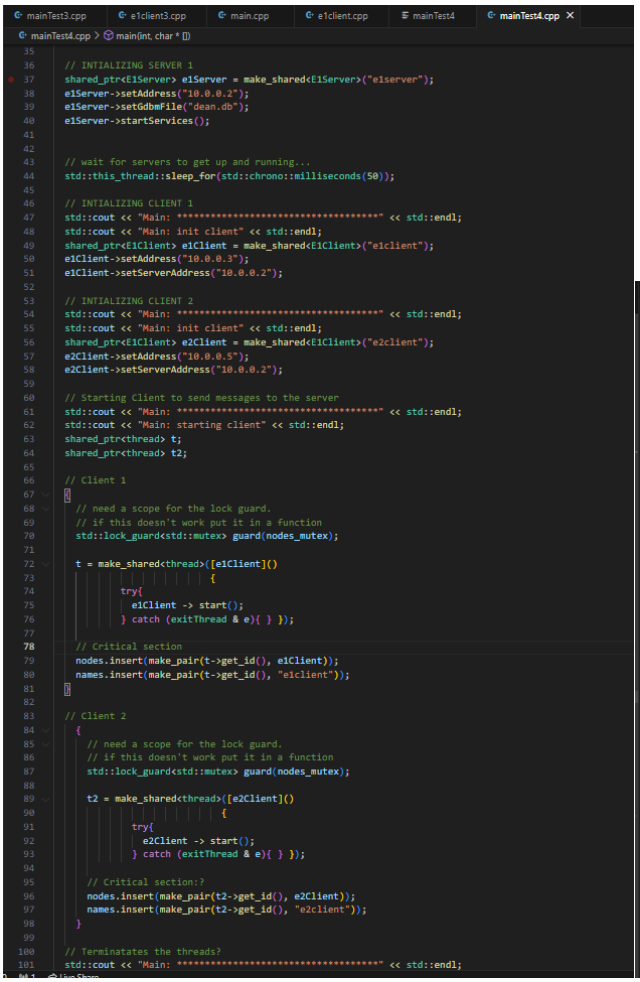
4. You should have tests that write different values to the same server with the same key

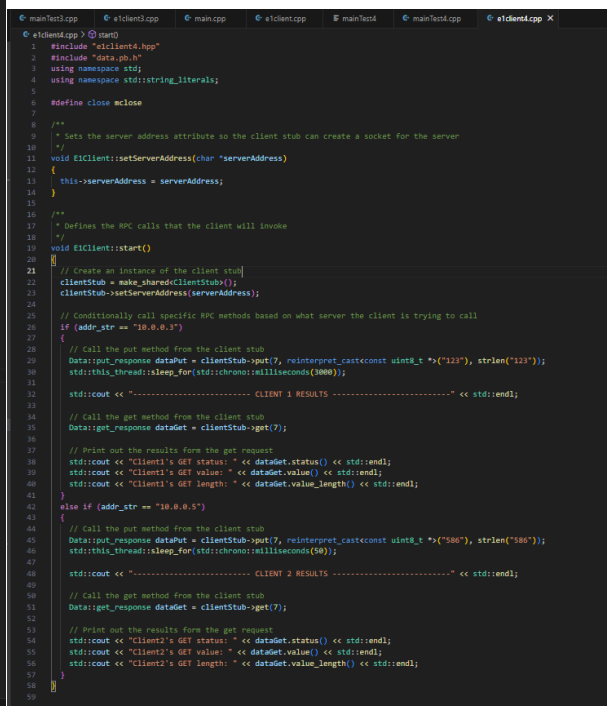
to overwrite, this test should be timed so that if one of the packets is delayed, the

value returned at the end by one of the clients is different.

In Test 4, we have two clients that write two PUT requests and then call two GET requests. However, the first client’s GET request is delayed meaning that the PUT request from the second client will overwrite the value in the gdbm database. When both clients return their values, we see that the GET responses

print out the second client's stored value. Therefore the second client has overwritten the first client's initial value.





```
--------------- E1SERVICE: Put Request Finished ---------------
CLIENT STUB: Success status of the put request:0
------------------------- CLIENT 2 RESULTS -------------------------
E1SERVICE: server received 12 bytes
E1SERVICE: Client Address 10.0.0.5
--------------- E1SERVICE: Get Request Finished ---------------
Client2's GET status: 1
Client2's GET value: 586
Client2's GET length: 8
------------------------- CLIENT 1 RESULTS -------------------------
E1SERVICE: server received 12 bytes
E1SERVICE: Client Address 10.0.0.3
--------------- E1SERVICE: Get Request Finished ---------------
Client1's GET status: 1
Client1's GET value: 586
Client1's GET length: 8
Main: **********************************
Main: calling stop services on server
Main: **********************************
Main: waiting for threads to complete
Main: shutting down protobuf library
```