ELEC 477

Kevin Yu: 20203451

Raatik Sharma: 20120770

Ainsley Taylor: 20210012

2024-02-05

Clientstub

The clientstub.cpp is where the client stub functionality is defined for the RPC protocol. It implements the lower-level procedure call for a key-value store communication system with PUT and GET RPCs. First, the UDP socket is configured using the "createSocket" method. It sets up the server's address and a timeout and initializes the socket configuration. Next, a PUT method sends a key-value message to the server where the key is used to specify the location of the data stored in the gdbm database. Also, the method manages valid message IDs and proper timeouts. A GET operation does the same process as PUT, but the key sent to the server in the key-value message fetches the corresponding data associated with that key in our gdbm database. We also set up a "closeSocket" method that closes the UDP socket. Many aspects of the code handle errors, such as a mechanism for checking for failed socket creation and serialization errors.

E1client

The e1client files are used to simulate clients in the RPC protocol. The e1client.cpp file implements an instance of the ClientStub. The method does a PUT operation for the value "kevin\0yu" with a key of 7 and a length of 10. If the method is successful, it prints out a success message, and if it fails, it prints out a fail message. The e1client.cpp file then performs a "get" method with a key of 7. It then prints the value and length from the "get" and whether it failed or succeeded. There are several e1client files, and each file is used to simulate different hard-coded RPC actions called by the client. E1client3 and e1client4 are hard coded to perform specific RPC actions depending on the client's address.

E1service

The e1service.cpp file simulates the server stub in the RPC protocol. There is a "setGdbmFile" method that is used to set the name of the file, which will store the key-value data. There is then a "stop" method where, when called, the server closes both the GDBM and the socket. We next implemented a "start" method that sets up the socket and binds it to the specified port. Throughout, there are error handling implementations for if the socket creation fails or the bind fails. The "start" function opens the GDBM database and handles incoming messages. When a message is received, the header is checked, and the type is checked and handled in the "dispatch" method. The "put_request" method first ensures the GDBM file is accessible and the database is properly being pointed to, then updates the value of the key_value_message in the GDBM file and sends back a serialized response based on the "put" operation's outcome. Similarly, the get_request method ensures the GDBM variables are valid and then accesses the key to fetch its related value in the database. If a value exists, the response is then serialized with the "true" status.

E1server

The e1server.cpp file simulates the server(s) in a RPC protocol. The code starts with including various libraries necessary for e1server.cpp to function. The constructor requires a string parameter called nodeName to specify the server's name. Within the constructor, an instance of E1ServiceServer is created, and the addService() method adds the generated service to the list of services in the node class for management. The setGdbmFile() method takes the string parameter called gdbm_name to set the gdbm file name in the service (server stub) class.

Main

The main.cpp file sets up a client and server to simulate a single RPC service. The code starts off by including various libraries that are necessary for main.cpp to function, followed by initializing "e1server" with an IP address of "10.0.0.2" and a GDBM of "dean.db". The server is set before the client to allow the server to do its setup configuration. The client "e1client" is then initialized with its own IP address of "10.0.0.3" and with the known IP address of "e1server". A thread for the client is started separately from the server to allow the server and client to execute in parallel so they can do requests and responses. The lock-guarded critical section is necessary to prevent race conditions from occurring with the global "nodes" and "names" variables. After that, main.cpp waits for the client thread to finish by calling the join() method. Then, the stopServices() method is called to stop the service (server stub) instances to allow the servers to eventually shut down. Following this, the waitForServices() method is called on the "e1server" to wait for the threads related to "e1server" to finish. Subsequently, the protobuf library is correctly shut down, and int main returns to 0.

Data.pb

This file is created from the protocol buffer compiler. It is used for serializing and deserializing the data into the respective data structs for our key-value messages used by our clients and servers.