Tonghui Ma, Ruijie Zhang
20376486, 20487924
July 12, 2017

CS 454 Assignment 3

Program Documentation

Part 1: File Contents

We have the in total 27 files for this project, they are:

Makefile

Documentation.pdf

binder.cc

binder_db.cc

binder_db.h

client.c

constants.h

error.h

helper.cc

helper.h

message.cc

message.h

message_handler.cc

message_handler.h

procSignature.cc

procSignature.h

rpc.cc

rpc.h

server.c

server_function_skels.c

server_function_skels.h

servinfo.h

server_functions.c

server_functions.h

server_sock_db.cc

server_sock_db.h

We have 8 .cc file that we implemented, and have 6 corresponding .h files. The rest of the code files are from the original project init code.

Part 2: How to run(see Part 5 about bouns)

1. use command "make" to create 3 executables "client" , "server", and "binder"
2. The binder needs to run first, to run, use ./binder

   it should print two statements, one is the BINDER_ADDRESS of which the binder is running on.

   the other is dynamically allocated port number BINDER_PORT

3. Set environment variable on both server machine and client machine using the following commands:

   export BINDER_ADDRESS=<get from binder output>

   export BINDER_PORT=<get from server output>

4. Launch server using command:

   ./server

   the server will try to register 4 functions to binder and waiting for client to connect.

   if something is wrong, appropriate error code will be returned.

   see the Error code section in this doc for details
5. Launch client using command:

   ./client

the client will first send location request message to the binder to get the server and port information about the server which provides this service.
Appropriate status information will be printed to the screen
if the previous action is successful, then the client will talk to the server
to call the remote procedure that it needed.

if any of the previous step encounters an error, then an appropriate error code will be returned and printed.

see the error code section for details

Part 3: Error code and Explanation

there are **19 constants** defined to indicate what is going to in the systems(errors and warnings and success indicator)

```
const int TYPE_ERROR = -1;
const int INVALID_LENGTH_ERROR = -2;
const int PROCEDURE_NOT_FOUND = -3;
const int NETWORK_ERROR = -4;
const int SEND_FAILURE = -5;
const int INVALID_BINDER_ADDRESS = -6;
const int INVALID_BINDER_PORT = -7;
const int CREATE_SOCKET_FAILURE = -8;
const int UNKNOWN_HOST = -9;
const int CONNECTION_FAILURE = -10;
const int ADDRINFO_FAILURE = -11;
const int BIND_FAILURE = -12;
const int LISTEN_FAILURE = -13;
const int MSG_NOT_RECEIVED = -14;
const int UNEXPECTED_MSG_TYPE = -15;
const int NO_REGISTERED_PROCEDURES = -16;
const int REGISTER_FAILURE = -17;
const int SEND_EXECUTE_MSG_TO_SERVERS_FAILURE = -18;

// const represnt register success or warning
const int REGISTER_SUCCESS = 0;
const int FUNCTION_ALREADY_EXIST = 1;
```

The name provides some information about each variable, now the detail information

TYPE_ERROR=-1:

When the receiver is expecting a particular type of data, but the sending sends data in another data type. E.g: receiver thinks the next chuck should be a string, but the sender says it wants to send an integer.

INVALID_LENGTH_ERROR = -2:

When the sender says an array of data, and before sends it, says the data will be in a Certain length. If the length is an invalid number (<= 0).

PROCEDURE_NOT_FOUND = -3:

The client requested a procedure, and that procedure is not found in the binder database Then this error will be returned to the client.

NETWORK_ERROR = -4:

This error means something wrong in the network, which usually means recv call failed due to some reason.

SEND_FAILURE = -5:

This error means that send() call failed. Usually due to network problem.

INVALID_BINDER_ADDRESS = -6:

This error means that binder address is invalid. Usually caused by not setting Environment variable correctly.

INVALID_BINDER_PORT = -7:

This error means that the binder port is invalid. Usually caused by not setting Environment variable correctly.

CREATE_SOCKET_FAILURE = -8:

This error is caused by failing create socket.

UNKNOWN_HOST = -9:

This error is caused by the cannot resolve the hostname.

CONNECTION_FAILURE=-10:

Caused by failed to connect. Could be network issue, or passed incorrect address and

ADDRINFO_FAILURE =-11:

Create ADDRINFO structure failure.

BIND_FAILURE =-12:

Caused by binding the socket fail.

LISTEN_FAILURE = -13:

Caused by error in calling listen() function

MSG_NOT_RECEIVED=-14:

Caused by expecting an message, but fail to receive it.

UNEXPECTED_MSG_TYPE=-15:

Causing by expecting one type of message, but receive a different type.

NO_REGISTERED_PROCEDURES=-16:

Causing by server cannot find the procedure which the client is requesting.

REGISTER_FAILURE=-17:

When the server fail to register a procedure in the system

SEND_EXECUTE_MSG_TO_SERVERS_FAILURE=-18:

Happens only in rpcCacheCall, when all the server cached functions have been tried and none of them work.

REGISTER_SUCCESS = 0:

The binder responds to the server to let the server know the register has succeeded.

FUNCTION_ALREADY_EXIST = 1:

Warning message that let the server know that the registration has succeeded, but the

Part 4 Message types and explanation:

R_Message

Server send this to the binder to register a procedure.

R_S_Message

Binder responds to server if register succeed with a warning code

R_F_Message

Binder responds to server if register failed.

LOC_R_Message

Client sends to binder requesting information about a procedure

LOC_S_Message

Binder sends to Client to let it know that the location of the server who owns that procedure.

LOC_F_Message

Binder sends to Client to let it know that the request of finding a procedure has failed. It contains an error code inside to let the Client know the reason of failure.

E_Message

Client sends to the server to ask it to execute some functions and with the arguments contained inside.

E_S_Message

Server sends to Client to let it know the request has succeeded with the execution result inside.

E_F_Message

Server sends to Client to let the Client know that execution has failed with error status inside.

C_LOC_R_Message(Bonus)

Client sends to Binder, requesting the Location of the servers of a particular function and cache them all.

Part 5: How to Run Bonus:

The bonus function has been implemented and **does not** need a different make/compile process.

To test the rpcCacheCall, simply write a new client.c that calls rpcCacheCall. This client can still calling rpcCall and all other functionalities.

Part 6: Design choices and implementations:

Data Marshing/unmarshing:

We defined several Structures to serve as different types of messages. These structs are defined in message.h and implemented message.cc. These messages are served as Registration Message, Location Request Message and etc. There are **send and read** methods defined for different messages, and they will read and send on each end. During implementation, on the sending end, we first send a 4-byte that indicates what type of message it is, and then send the content. On the receiving end, we first read the 4 bytes to know what type of message it is, and then call the corresponding read method to read the rest of the content.

Handling function overloading:

The binder database is implemented as a vector. Each element in the vector contains the registration message. So every time we insert a new message, we will loop through the vector and check if it is a new function or it is the same function from the same server registers twice. Besides just comparing function names, we implemented a helper function that compares two argTypes array and check if they are exact match. We used the comparison logic specified in the assignment to implement this(used bit shift operations to do it). So we will distinguish between data types, arrays and scalars. So after the through comparison, **we only overwrites if it is the exact same function that comes from the exact same server, otherwise it is added to the database.** So we can definitely handle function overloading, and handle different servers register the same function.

Round Robin:

The binder database is implemented as a vector. So each time we selected a server address from the database. After sending server address to the client, this address is moved to the back of the vector. Each time we loop through the vector, we loop from the beginning. So make sure the round robin by using the fact that vector data structure preserves order.

Termination Message:

The binders have another database that stores how many servers have registered. So each time if a server registers, it will check if it is a new server, if yes, it will add that to the database.

When the binder receives the Termination Message, the binder will then send the termination message to each server. And the servers will then terminate because of receiving such message.

All functionalities have been implemented including the bonus functionality.