Lawson Fulton (ljfulton) 20381453
Devruth Khanna (dkhanna) 20295702

System Manual

Marshalling/Unmarshalling of Data

- This is handled in the Message class. Marshalling is handled by the “serialize” functions, while Unmarshalling is handled by the “deserialize” functions.

- Serialize: Based on the message type (REGISTER, LOC_REQUEST, LOC_SUCCESS, EXECUTE, EXECUTE_SUCCESS, REGISTER_SUCCESS, REGISTER_FAILURE, LOC_FAILURE, EXECUTE_FAILURE, TERMINATE), a message is marshalled. Since different parameters depend on the type of message, there are functions that accommodate the marshalling of strings, integers, and ArgTypes (which specifies the input and output parameters).

Serializing of Args Array

The args array is serialized differently depending on whether it is being sent from the client to the server, or from the server to the client. In the client to server case, only args with corresponding arg type including ARG_INPUT are serialized. In the server to client, only ars with arg type including ARG_OUTPUT are serialized. This saves bandwidth as no unnecessary information is transmitted.

- Deserializing:

Deserializing the args array on the server side is straight forward. Memory is allocated for the appropriate variables, and the received data is copied into that memory.

On the client side it is more complicated since args labeled as ARG_OUTPUT must have their data copied into pre-existing memory locations on the client. This is achieved by keeping the original args array that the client submitted, and then copying data from the deserialized array into the memory locations listed in the original args array.

Structure of Binder Database

Briefly, the binder database is a mapping between function signatures and sets of server addresses that implement that function. When the binder receives a LOC_REQUEST, it can simply create a FunctionSignature object from the provided name and argtypes. The binder can then query the database with that signature to obtain a set of ServerInfo objects, where each ServerInfo stores the address and port of a server process that implements the given function.

- The Database is defined as:

```
std::map<FunctionSignature, std::set<ServerInfo>> database;
```

- The binder database is defined as a map, where the key of the map is the class FunctionSignature, and the value is a set of ServerInfo classes.

- The FunctionSignature class defines the registered methods of the form: (char* name, int* argTypes), where “name” defines the name of the registered method, and “argTypes” defines the types of the input and output parameters. ServerInfo defines available servers, defined by the address and port number.

- A map was the ideal datastructure to use, as a lookup can be done in constant time, as a map is really a hashtable.

Lawson Fulton (ljfulton) 20381453
Devruth Khanna (dkhanna) 20295702

Handling of Function Overloading

Function overloading was a natural result of making the database key a FunctionSignature object. Since the < operator for the FunctionSignature class compares on the basis of function argument types, as well as the function name.

Managing Round-Robin Scheduling

- Round robin scheduling is done in Binder::sendLookupResponse. In the database mentioned above, which maps FunctionSignatures to a set of ServerInfo objects, there is a member variable that is a timestamp.

- At the time that a ServerInfo object is created in the constructor, the timestamp is initialized. Whenever a particular method is executed, its respective ServerInfo object is referenced, its timestamp is updated as the current time.

- Whenever there is more than one ServerInfo object registered for a method, the ServerInfo object with the smaller timestamp value (i.e the older timestamp, which would have a smaller timestamp value), would be executed first.

- This process of selecting the server that has gone the longest without executing a function effectively implements the Round Robin scheduling algorithm.

Termination Procedure

The system can be terminated in a variety of ways. First, the most straight forward way is by the client calling rpc_terminate. This will send a termination message to the binder, which will in turn send a termination message to all of the servers and shutdown. The servers will exit immediately upon receiving a termination message from the binder.

The second way this can be accomplished is by terminating the binder directly. The servers are continuously monitoring their connection with the binder, and upon detecting the connection closing, will automatically terminate themselves.

Each server could also be terminated independently. When the binder detects that a server has disconnected, that server is removed as a location for any functions it has registered. If it was the only server for a given function, that function is removed entirely.

Error Codes

- Error codes are defined in MyExceptions.h:

```
CONNECTION_ERROR = -1,
        // Error connecting to a socket

ENV_VARS_NOT_SET = -2,
        //Environment variables (Binder_Address and Binder_Port) were not set

RECEIVE_HEADER_ERROR = -3,
        //Header of message was not received

RECEIVE_BODY_ERROR = -4,
```

Lawson Fulton (ljfulton) 20381453
Devruth Khanna (dkhanna) 20295702

```
        //Body of message was not received

SEND_ERROR = -5,
        //Error with sending a message

UNKNOWN_HOST_ERROR = -6,
        //Error with trying to lookup host

SOCKET_ERROR = -7,
        //Error in creating a socket


DISCONNECTION_ERROR = -8,
        //Error in trying to disconnect

UNEXPECTED_MESSAGE_ERROR = -9,
        //Reveived an unexpected message

SERVER_FUNCTION_NOT_FOUND = -10,
        //Errror - Specified server function is not specified/registered

BINDER_FUNCTION_NOT_FOUND = -11,
        // Error - binder function not found

RUNTIME_LOGIC_ERROR = -99999;
        //Never occurs in normal execution, specifies inconsistent program state.
```