# CS454 A03 System Manual


## Developers:

Ji Ho Park (20518866): jh49park@uwaterloo.ca
Myungsun Jung (20511678): m22jung@uwaterloo.ca


## System Implementation Details

### Marshalling/Unmarshalling of data:

In order for the rpc_client, binder, and rpc_server to make any network calls, they have to use sending functions that are defined in the message_lib.cpp. In these functions, by taking arguments like name, argTypes, args, etc, it initially calculates the size of the message. Based on the result, a buffer of character array is defined to copy all the argument data into the array. The message length calculated and the message type is also copied into the buffer and then get sent through socket.
For the args, value of each arguments in args are received by de-referencing them. These retrieved arguments are put into the buffer in a different way depending on the type of the argument. For example, for an integer argument, the value of it is cut into byte by byte and put into a buffer. Since size of integer is 4 bytes, the message pointer increments by 4.
Unmarshalling received data is pretty much the same process in same order. First, it extracts length and msgType from the received message. Then based on the msgType, the system figures out which elements to extract from the message. For example, if the msgType is RegSuccess, the system knows that there will be a server_identifier and port marshalled in the message. Hence, it extracts these two elements based on their size.
These extracted, or unmarshalled, data gets passed into FunctionData constructor in order to further unmarshall detailed argTypes and args. These data is either saved in local database or gets used in different purposes.

### Structure of binder database

The local database of the binder is implemented using a vector. The element type in the vector is a custom structure called ServerData. The structure contains basic server information like hostname and port number. In addition, ServerData has a vector of custom class called FunctionData. FunctionData class is basically the same manner as ServerData, which holds information regarding on remote functions. As rpcRegister message comes in from the server, the system checks whether a FunctionData constructed based on the sent name and argTypes exist in the database. If there is an existence, then the binder sends back a RegSuccessMessage back to the server with a reason code indicating that there is a duplicated function definition in the binder. Otherwise, it simply adds the new FunctionData to the corresponding vector.

Handling of function overloading

In the local database of the server, there is a vector of custom class that basically saves supporting function's name, argType, and corresponding function skeleton. As rpcRegister() is called, the system first checks if there is any duplicated instance in the database. If there is, then the system updates a function skeleton of the matching database instance. In the case of overloading, rpcServer does not make a network call to the binder because the binder does not need to know anything about the function skeleton.

Managing round-robin scheduling

When a location request comes in, an iterator loops through the local database in a round-robin queue manner until it finds a server that supports the requested function skeleton. Instead of creating a new queue by copying list of servers from the database, the system uses a single global pointer and the pointer iterates through the existing list. Since our database is implemented in a server manner (i.e. elements within the vector is in form of ServerData), the database itself can be used as a queue.

Termination Procedure

In order to terminate the system, a client executes rpcTerminate() function. It sends the request to the binder, and the binder will inform the servers by passing the request to the registered list of servers. In each server, the server verify that the termination request comes from the binder's IP/address for the sake of simplicity. After the authentication, servers terminate and the binder terminates after all servers have terminated.

Error Codes

REG_SUCCESS_NEW_SERVER = 1,
- Registration of new server succeeded

REG_SUCCESS_EXISTING_SERVER = 2,
- Registration of existing server succeeded

REG_SAME_FUNCTION_EXIST = 3,
- Registration of same function succeeded

FAILURE = -1,
- Generic failure message

BINDER_ADDR_NOT_FOUND = -2,
- Binder address not defined in environment variable

BINDER_PORT_NOT_FOUND = -3,
- Binder port not defined in environment variable

BINDER_NOT_SETUP = -4,

- Binder Socket not properly set up to make a connection

SERVER_SOCKET_NOT_SETUP = -5,

- Server Socket not properly set up to make a connection

SOCKET_NOT_SETUP = -6,

- Socket not properly set up to make a connection

SOCKET_CONNECTION_FAILED = -7,

- Failed to make a new connection to a socket

SOCKET_CONNECTION_FINISHED = -8,

- Socket connection has been terminated

SOCKET_OPENING_FAILED = -9,

- Opening a new socket failed

DATA_SEND_FAILED = -10,

- Sending data over socket failed

READING_SOCKET_ERROR = -11,

- Reading from a socket failed

FUNCTION_SKELETON_DOES_NOT_EXIST_IN_THIS_SERVER = -12,

- Requested function does not exist in the server

FUNCTION_LOCATION_DOES_NOT_EXIST_IN_THIS_BINDER= -13,

- Requested function does not exist in the binder

NO_HOST_FOUND = -14,

- No such host found

ERROR_ON_LEASON = -15

- Error occurred while listening to a socket

Unimplemented Features:

Bonus feature has not been implemented. Other than that, all the required functionalities have been implemented.