Documentation

1. **(un)marshalling data –** To create, read and write messages for sending, we created an API in common/message.h. To construct a message all that must be done is to consecutively call the read/write functions for the base message types (string, int, argTypes, args). The message structure contains a char* pointer to the message, the allocated length (for dynamically resizing the message if the allocated space is not enough), and the length (which is the current number of bytes in the message that will be sent over the wire).

   Binder database/Function Overloading – To store the function signatures, we first mapped the function names to trees defining the function signature. To insert a function into the tree, we take the argTypes and use these as keys for each layer in the tree. Once we have inserted all the argTypes into the tree we put a "leaf" (by this we mean it is a node corresponding to a valid function signature), which is the server address/port. To look up a function from the signature, all we have to do is traverse the tree and see if there are any "leaves" on the final node.

   Round-robin scheduling – The aforementioned "leaves" (see above paragraph) are actually stored in a vector, so we can have multiple servers with the same signature. Each time a "get" operation is done for a particular function signature, we use an index to choose a server, then increment and wrap the index.

   Termination procedure – Unfortunately we overlooked this, and had to hack a way of listening for a TERMINATE message from the binder. We create a thread in each server to specifically listen for this message. We did this on account of having designed everything single-threadedly and the "select" function blocking our thread.

2. 0 NONE
   1 WARNING_DUPLICATE_FUNCTION – When a function has been registered to the binder with the same function signature
   -1 FAILED_SEND – When send() has failed
   -2 FAILED_RECEIVE – When recv() has failed
   -3 FAILED_BINDER_CONNECT – When a connection to the binder did not succeed
   -4 FAILED_SOCKET_BIND – When creating a connection socket fails
   -5 FAILED_SOCKET_LISTEN – When listening to a connection socket fails
   -6 INVALID_SOCKET – When an invalid socket has been used
   -7 INVALID_FUNCTION_SIGNATURE – When a function with the specified signature hasn't been registered, but has been called by rpcCall
   -8 FAILED_FUNCTION_CALL – When a rpcCall has been made, but the procedure fails.

3. Blocking calls will block the server until they have been finished (ie. No multithreading)

4. Nothing.