Documentation and System Manual

1. **Design Choices**
    a. Marshalling and Unmarshalling data
        - Data pertaining to arguments is marshalled by converting it to a char array format and then sending that over.
            1. rpcCall(...), rpcRegister(...) marshalls arguments by converting them into a string format: ie. “LOC_REQUEST;add;int*,int;”.
            2. rpcCall(...) marshalls arguments by copying over their bytes into a char array and then sending that over to the server.
        - Unmarshalling data involves reading an integer, n, denoting # of bytes, and then reading n bytes into a char array. Given the context, that char array can be appropriately casted to an array of another type.
    b. Structure of Binder Database
        - Dictionary of form Map< String, Set < Pair <String, String >> >.
            1. Key is the concatenation of function name and arg types, ie “add;int,int*;”.
            2. Value is a set of pairs, in which each pair is of form <server id, port number>.
        - When a server sends a register request to the binder, it sends the key, server id, and port number, and the binder creates/appends the entry in the Binder Database.
        - When a client sends a location request to the binder, it sends the key, and the binder, if possible, returns a pair found in the corresponding set.
    c. Handling of Function Overloading
        - Since the key for the Binder Database dictionary is a concatenation of a function name and its arg types, ie “add;int,int*;”, the handling of function overloading occurs naturally.
    d. Managing round-robin scheduling
        - Active servers are stored in a list of form List< Pair <String, Int > >.
            1. The String is a concatenation of server id and port number.
            2. The Int is the corresponding server’s socket file descriptor.
        - To implement round-robin scheduling
            1. Search in the active servers list in order for the first Pair<String, Int> that can service the given function. Retrieve that Pair.
            2. Parse the String and return the relevant server info to the client.
            3. Remove the Pair<String, Int> from its current position in the list, and place it at the very end. Now, another server providing the same service can be selected next by step 1.
    e. Termination Procedure

- Active servers are stored in a list of form List< Pair <String, Int > >.
    1. The String is a concatenation of server id and port number.
    2. The Int is the corresponding server's socket file descriptor.
- When a client sends a termination request to the binder, the binder sends a termination request to all servers stored in the active servers list. Once each active server has acknowledged the termination request by returning a message to the binder, the binder terminates.
- Since the server only terminates if the termination request originates from the binder, binder authentication is assured.

2. Error Codes

a. rpcInit()
- -1 if client connection cannot be established.
- -2 if binder connection cannot be established.

b. rpcRegister(...)
- 1 if given function has already been registered
- -1 if rpcInit() has not been called
- -2 if binder is not located
- -3 if binder error - protocol not followed

c. rpcExecute(...)
- -1 if rpcRegister(...) has not yet been called
- -2 if select(...) failed
- -3 if server cannot connect to client

d. rpcCall(...)
- -1 if client request does not adhere to protocol
- -2 if binder connection cannot be established
- -3 if given function is registered but no server available
- -4 if server connection cannot be established
- -5 if given function is not registered, therefore unavailable
- -6 if given function's execution failed server-side

e. rpcTerminate()
- 1 if binder and servers already terminated
- -1 if binder connection not established

3. Functionationality that has not been implemented

a. rpcCacheCall(...) has not been implemented.

4. Advanced Functionality

a. N/A.