System Manual

General Design and Data Marshalling

Firstly, an abstraction was created called a Connection. A Connection sends and reads messages over a socket, where all messages are at least 8 bytes in length. The first 32 bits signify the length of the message in bytes (excluding the first 8 bytes), and the second 32 bits signify the type of the message.

The Connection class then exposes an API to send a type and a string, as well as an API to read into a type and string. Now, all data must simply be serialized to a string in order to be passed over the network.

A class was created called FunctionSignature, consisting of a name and a vector of integer argument types. An equality operator was defined on this class to compare both the name and the types, ignoring the magnitude (but not the presence of) arrays. This equality operator then transparently handles function overloading, so no additional logic is required. The FunctionSignature class includes logic to determine the size, in bytes, of a given argument.

Another class was created called FunctionCall, consisting of a FunctionSignature and a vector of bytes. When this class is initialized with the argTypes pointer and the void** data pointer, it uses the FunctionSignature size logic to copy the data from the void** data pointer into its own data vector. The data then resides sequentially in the FunctionCall's vector, so it is easily serialized and deserialized.

A third class called ServerAddress handles serializing an address consisting of a port and a hostname. The hostname is assumed to be less than 100 bytes long (although this is a hardcoded constant it could easily be changed to adapt to longer lengths).

Binder Database and Round Robin

In the interest of simplicity and reliability, the design of the binder's database was kept intentionally simple. The binder's database is a basic std::list of Server objects. A Server object consists of a ServerAddress, a socket file descriptor and a list of FunctionSignatures. In this list, the socket file descriptor is used as the key.

When a server registers a function (as a serialized FunctionSignature), the server list is searched for the matching socket, and the signature is appended to its list of FunctionSignatures. When a client requests a server to handle a particular FunctionSignature, the list is searched from the beginning, and the first server with a matching FunctionSignature is returned.

This design makes the round-robin implementation very straightforward. When a server is found for a client's request, the server is moved to the end of the binder's list of servers. It is not the most efficient approach, but should be easily performant at the scale required for this assignment.

An alternative design is possible, but the round robin approach as prescribed makes any alternative design challenging (since the round robin operates globally across all servers, but a FunctionSignature applies only to a limited set of servers). A description follows, but may be beyond the scope of this document. If a hash function were added to a FunctionSignature then it could be used to index a set of ServerAddresses, and both registration and lookup would be “fast”. The selected set would then be compared to a global queue (linked list) of servers to find the first occurrence, which would then be moved to the back. Of course, removing a server would be more time consuming, but that is (probably) a rare occurence.

Error Codes

Distinct error codes were not used. A warning was used to indicate when an overriding (not overloading) function was registered, but the registration proceeds regardless.

Missing Functionality

To the best of my knowledge, no functionality is missing.