Structure of binder database / Round-robin scheduling:
All data is stored in a map where keys are procedures and values are server locations stored in a queue. When a procedure is called, the corresponding queue is found and the server location at the front of that queue is popped and pushed to the very end of the data structure, thus achieving simple round-robin scheduling.

Function Overloading:
All procedures are stored in a map which allows for function overloading. Functions with similar names but different arguments are stored as 2 separate functions. This is achieved by overriding the less operator in the Procedure class, which is used as the keys for the database. When comparing 2 procedures, the less operator function takes each Procedure's name concatenated with its argtypes and alphabetically compares them. This means that 2 procedures with differing argtypes can exist separately and selectively called when needed.

Termination procedure:
When the client calls terminate, it sends a client-binder message to the binder, then a terminate message to let it know that it wishes for it to execute the terminate procedure. The binder then goes through a list of server locations (stored in a queue) built from every rpcInit called by each server and through protocol sends them request messages. Once all the request are sent, the binder closes all its remaining sockets and finishes execution. All the servers check the IP of the binder before doing the same.

Marshalling/Unmarshalling of data:
When rpcCall is called, the argTypes array is sent to the server so that it knows how to interpret subsequent data. Each element of the argTypes array is then analyzed to determine the transfer protocol from client to server. The input/output bit, type name, and argument length of each argType element is unmarshalled and stored into an int. Once this metadata is extracted, the client begins marshaling the parameters. The client uses the length of each argument and sends them one by one.
The server then receives the arguments per protocol and marshals them into the server skeleton which is obtained through the local database modelled by a map structure. The result is returned within the inserted args array. The server then simply marshals the results to the client per protocol.

Error Codes:
-1: ERROR_BINDER_NOT_SET;
    The BINDER_ADDRESS and/or BINDER_PORT environment variables have not been set.
-2: ERROR_SOCKET_CREATION_FAILED;
    An error occurred during socket creation in rpcInit or in the listening socket in the binder.
-3: ERROR_LOC_DATABASE_NULL;
    The procedure-location database has not been created successfully.
-4: ERROR_LOC_NOT_FOUND;

A valid server to execute the RPC call has not been found.

-5: ERROR_EXECUTE_FAILURE;

Execution of the RPC call has failed on the server.

-6: ERROR_SOCKET_READ_FAILED;

An error occurred when reading a message from a remote host.

-7: ERROR_SOCKET_WRITE_FAILED;

An error occurred when writing a message to a remote host.

-8: ERROR_SOCKET_ACCEPT_FAILED;

An error occurred when accepting data from a remote host.

-9: ERROR_SOCKET_CONNECT_FAILED;

An error occurred during connection to a remote host.

-10: ERROR_SKELETON_NOT_FOUND;

A skeleton for the specified procedure has not been found in the RPC local database.