CS454 Assignment 3 Documentation

Chen Huating (h255chen)
Fu Jingge (j35fu)

Our RPC library implements all the standard assignment specifications (no bouns)

Software structure:

- RPC and sockets
    - Relative file: clientsocket.cc serversocket.cc rpc.cc
    - clientsocket.cc: Creating client socket which is responsible for connection to server and binder.
    - serversocket.cc: Creating server socket which is responsible for connection to client and binder
    - rpc.cc: rpc.cc uses serversocket.cc and clientsocket.cc to make connection. Message parsing and processing is done inside rpc.cc.
    - All the data can be divided into 3 parts: length, type and data. Use memcpy to convert all the data into char array. The receiver will get the length and type first and select different procedure to process the data according to the type.
- Binder
    - Relative file: binder.cc
    - Binder receive register requests or location request from server and client.
    - Upon the arrive of a register request, the binder create two structs: a `ServerInfo` contains the infomation (hostname and port) about the server, a `ProcedureInfo` contains the function name and function argument type. Then the binder will search through the database for the server and the function.
    - By overloading the comparsion operator of the two structs, the binder can handle the overloading of function. For `ProcedureInfo` , two function is consider the same when they have the same name and same argType (might have different array length). For `ServerInfo`, the name and port must be identical.
    - Inside the database, there are two type of data, server information (`ServerInfo`) and function information (`ProcedureInfo`). Each of them is a strcut and is stored in a table. In `ServerInfo`, it has a list of

`ProcedureInfo*` that it registered, and in `ProcedureInfo`, there is a list of `ServerInfo*` which registered this function. Everytime a location request is arrived, it first finds the function in the database, then return the first `ServerInfo` in its list, then move the first `ServerInfo` to the last. In this way we have a round-robin order.

- Uility function:
    - Relative files: constants.cc io.cc
    - constans.cc provides some function which can be used for message parsing. Includeing function like `CharToInt()` which takes a char array of size 4 and return the correspoding integer.
    - io.c provides interface for message sending and receving. The user only need to provide the type of the message and the message buffer, io.c will compute a new buffer according to the protocol and send it, or receive a message and fill in the buffer. io.c also handles the case where the buffer is too large and `send()` and `recv()` can only handle part of the message.
    - The marshall and unmarshall of a message is all done in io.c. The protocol we used is exactly the same as section 5 in the assignment.
- Termination
    - The client will send a termination message to binder .Once binder get the termination message, it will send the termination message to all alive server and wait for their response one by one. All servers will send the response and quit after receiving the termination message. After receiving all responses from servers, binder quits. (We suppose the rest of clients will quit gracefully)

Error code:

(All error code are defined in constants.h)

- ERRORBINDERINFO
- Not able to get the binder information

- ERRORBINDERCONNECT
- Not able to connect to binder

- ERRORSENDREG
- Error sending the register data

- ERRORSENDLOCREQUEST
- Error sending the loc_request

- ERRORSENDTERM
- Error sending termination message

- DBEMPTY
- The local database for server is empty

- NOSERVER
- Server does not exist

- WHAT
- This should not be print!

Functionality:

All functionalities of the assignment are satisfied

Advanced Functionality:

None