CS454 A3 Manual

Yi Kun Song 20385742 yksong
Hong Wen Zhu 20376734 hwzhu

1. Design Choices

1.1 Marshalling & Unmarshalling

A very simple but effective method is used for data marshalling.

Send:
For majority of messages, we first precompute and send integer size of entire message, then create a buffer that has the same size. Next, we use memcpy and some pointer arithmetic to copy message content to the buffer and send the buffer using sendAll function.
char/int * such as strings/argTypes are copied by first computing its size using ptrSize function including trailing null. And then copy them to the buffer.
On the other hand, EXECUTE and EXECUTE_SUCCESS are sent in a little different way because it contains void **. First, the size of message excluding void ** are sent, followed by message type, name, and argTypes. Then it loops through the void ** and send void * based on information stored in argTypes.

Receive:
For majority of messages, we receive the size of message first, then allocate a buffer of that size; Then we receive the entire message and then do some pointer arithmetic to copy from receive buffer to local variables.
char/int * are received in a similar manner.
On the other hand, EXECUTE and EXECUTE_SUCCESS messages are received in the way they are sent: we first receive the size of message type, name, and argTypes, then receive args based on information in argTypes.

1.2 FunctionDB

A simple FunctionDB class is created for handling signature lookup and round-robin scheduling. FunctionDB actually wraps a map object with very simple interfaces:

```
struct ServerInfo locate(struct ProcedureSignature);
void register_function(struct ProcedureSignature, struct ServerInfo);
```

register_function is used for registering functions when binder gets the function signature and serverinfo from the server. locate is used from client's location requests. These two methods are actually put and get with specific purposes.

Key-Value signature
std::map<ProcedureSignature, std::list<ServerInfo>* >

```
struct ProcedureSignature {
   std::string name;
   int *argTypes;
};
```

ProcedureSignature is a custom struct created for the specific requirements on function signature. It contains the function name and the argTypes. We want to use this struct as the key of the map so that the look up operation will be trivial. However, ProcedureSignature cannot be used as the key by default. A comparison function needs to be overridden.

In the comparison function, we first check if the function name matches. It the name matches, the argTypes array is iterated. As described in the spec, the type is only defined in first 16 bits, the rest of the int is actually the length. We will check if both argTypes[i] & 0xFFFF0000 is the same and if they length are both 0 or neither 0. During the iteration the length of argTypes is also checked.

For round-robin scheduling, we take the advantage of std::list. The nodes in the list can be easily manipulated. In our implementation, for each locate (look up), the first element in the list is popped, and then it is pushed into the back of the list.

1.3 Termination

The termination procedure follows the spec. Client sends a TERMINATE request to the binder through rpcTerminate(). In rcpInit, the server initiates a connection to the binder, and the connection socket with the binder is stored in a list. The binder uses the same connection to send the terminate request back to the server. For authentication, server needs only to check if the connection is the one with the binder. The server will then send the corresponding response message to the binder.

2. Error Codes

Error code are defined in common.h as follows:

```
enum ReasonCode {
    ENV_NOT_SET = -1,                        // environment variables are not set
    SEND_FAILED = -2,                        // socket cannot send successfully
    RECV_FAILED = -3,                        // socket cannot receive successfully
    UNKNOWN_MSG_TYPE = -4,                   // received message but type unexpected
    TERMINATE_CALL_NOT_FROM_BINDER = -5,     // terminate call is not sent from binder
    LOC_FAILURE_SERVER_NOT_FOUND = -6,       // cannot locate server
    SKELETON_FAILURE = -7,                   // skeleton failed to execute properly
    REGISTER_DUPLICATE = -8                  // another function of the same server exists
};
```

3. Not Implemented Functionalities

We've done it all except for bonus rpcCacheCall

4. Extra Info

N/A