CS454 A3 System Manual

Luke Leung-Toung
Kelvin Yang

Introduction

This manual describes at a high level the implementation of our distributed system which fulfils the CS454 Assignment 3 specification.

Section 1: Messaging Protocol

For the most part, we followed the protocol specified in section 5 of the assignment specification. However, rather than send the length of the entire message at the beginning, we send a length message before sending any piece of variable-length data. So for example, sending the server address and port “localhost” and “50100” involves sending 4 messages:

The number `10` (4 bytes)
The string `“localhost\0”` (10 bytes)
The number `6` (4 bytes)
The string `“50100\0”` (6 bytes)

This increases the number of messages sent within the distributed system, but makes the programming much easier, as we know exactly how much memory we need to allocate for each buffer.

Section 2: Implementation Details, RPC Functions

We only cover details relevant to the front end (i.e. Client and Server). Details relevant to the Binder will be covered in Section 3.

`rpcInit()`

This function is invoked by the Server to open up a connection with the Binder. The connection remains open until the Server receives a TERMINATE request.

1) [Server] Open 2 static sockets, one connecting the binder and one to listen for connections.
2) [Server] Send an INITIALIZE message to the Binder.
3) [Server] Create a pollfd array with the binder socket and listen socket.

`rpcRegister(char* name, int* argTypes, skeleton f)`

This function is invoked by the Server to register a function signature with the Binder.

1) [Server] Send the function signature `(name, argTypes)` to the Binder.
2) [Server] Add the mapping `(name, argTypes) -> (f)` to a database local to the Server.

`rpcExecute()`

This function is invoked by the Server to begin receiving remote procedure call requests

1) [Server] Listen to incoming requests.
2) [Server] Poll the sockets, checking for input.
3) [Server] If binder socket has a message, terminate.
4) [Server] If listen socket has a connection, fork a new process and accept the connection.

EXECUTE messages can only come through the listen socket. Therefore, the last step ensures that the Server can handle multiple calls without blocking.

`rpcCall(char* name, int* argTypes, void** args)`

This function is invoked by the Client to execute a remote procedure call.

1) [Client] Connect to the Binder
2) [Client] Send the function signature `(name, argTypes)` to the Binder.
3) [Client] Receive the response `(address, port)`, or an error if no Server available.

4) [Client] Connect to the Server at `(address, port)`.
5) [Client] Send the function signature `(name, argTypes)` and arguments `args` to the Server.
6) [Server] Look up the skeleton for that function signature from the local database.
7) [Server] Execute the skeleton, and return the result to the Client.

`rpcTerminate()`

This function is invoked by the Client to shut down the entire distributed system.

1) [Client] Connect to the Binder
2) [Client] Send the TERMINATE request to the Binder, then quit
3) Binder sends the TERMINATE request to each Server, then quits
4) [Server] Server quits

Because calls are handled by separate processes on the Server side, the Server quitting will not interrupt any calls in the middle of execution. However, new calls will be prevented from arriving.

Section 3: Implementation Details, Binder Specific

Handling of Function Overloading

The binder maps information in the form:

`(name, argTypes) -> Set< (address, port) >`

Since functions are stored using both their `name` and `argTypes`, it's possible to store multiple functions with the same `name` as long as the `argTypes` are different.

The set contains the `(address,port)` pair of each server that has the function with the `(name, argTypes)` combination.

Managing Round-Robin Scheduling

The binder has a list of all the servers connected to it called the queue.

Once a request from the client comes in, it retrieves the set of servers that can service this request from the database.

Then starting from the front of the queue, it checks each server to see if it can service the request. As soon as a server is found, it is selected to service the request and is moved to the back of the queue.

When a server comes in to register a function for the first time, that server is placed in the front of the queue. Therefore, newly added servers will always be at the front of the queue.

Server Termination

If the client has sent a TERMINATE message, the binder will then send the TERMINATE message to all of the servers connected to it before closing.

Section 4: Implementation Details, Server Specific

Handling of Function Overloading

Each server stores the functions it registers in a local database. The local database maps information in the form:

```
(name, argTypes) -> function
```

Since functions are stored using both their `name` and `argTypes`, it's possible to store multiple functions with the same `name` as long as the `argTypes` are different.

Marshalling and Unmarshalling of Data

When a client sends a message to a server, it is processed in the `rpcExecute` function and arrives in the form:

```
1) int EXECUTE
2) int nameLength
3) char* name
4) int argTypesLength
5) int* argTypes
6) void** args
```

The `nameLength` just makes it easier to read the name into an array. The `argTypesLength` is used to create an array of `serverArgs` which will hold a copy of an `argType` and its respective `arg`.

The server will then decode the `argTypes`, storing the length, variable type and a copy of each `argType` in the `serverArgs` array.

Using the information decoded from the `argTypes`, the Server will store the each `arg` into a pointer to a new variable of the correct type and size and put it into the `serverArgs` array.

Once all of this is done, the `argTypes` and void pointers are assembled into new arrays and passed into the skeleton function.

Server Termination

While the server is in its `rpcExecute` loop, it polls both its listening socket and its socket connected to the binder for input. If the server receives input from the, it assumes the input is the TERMINATE message and stops the server.

Section 5: Error Codes

-100	SIGNATURE_NOT_FOUND	Returned to Client when Client requests a function whose function signature was never registered
-300	TERMINATE_REFUSED	Returned to Client when Client attempts to send the `TERMINATE` request to a Binder that already received the `TERMINATE` request
-400	BINDER_CONNECTION_REFUSED	Returned to Client or Server when unable to connect to the Binder
-425	BINDER_DISCONNECTED	Returned to Client or Server when the Binder unexpectedly disconnects
-450	SERVER_CONNECTION_REFUSED	Returned to Client when unable to connect to the Server
-475	SERVER_DISCONNECTED	Returned to Client when the Server unexpectedly disconnects
-500	SERVER_NOT_INIT	Returned when Server calls `rpcExecute` or `rpcRegister` before `rpcInit`
-600	BINDER_ADDRESS_NOT_SET	Returned to Server or Client when `BINDER_ADDRESS` is not set
-700	BINDER_PORT_NOT_SET	Returned to Server or Client when `BINDER_PORT` is not set
-800	SERVER_REGISTER_FAILED	Returned to Server when the registration has failed in the Binder
-900	NO_REGISTERED_FUNCTIONS	Returned to Server when Server calls `rpcExecute` with no registered functions
100	SERVER_FUNCTION_ALREADY_REGISTERD	Returned when Server tries to register a function with the same `name` and `argTypes` as another already registered function