Design details for Assignment 3

1. Request code list

From	To	Message	TypeCode	Content
client	binder	LOC_REQUEST	1	LOC_REQUEST, name, argTypes
client	binder	TERMINATE	2	TERMINATE
client	server	EXECUTE	3	EXECUTE, name, argTypes, args
binder	client	LOC_SUCCESS	4	LOC_SUCCESS, server_identifier, port
binder	client	LOC_FAILURE	5	LOC_FAILURE, reasonCode
binder	server	REGISTER_SUCCESS	6	REGISTER_SUCCESS
binder	server	REGISTER_FAILURE	7	REGISTER_FAILURE
binder	server	TERMINATE	2	TERMINATE
server	binder	REGISTER	8	REGISTER, server_identifier, port, name, argTypes
server	client	EXECUTE_SUCCESS	9	EXECUTE_SUCCESS, name, argTypes, args
server	client	EXECUTE_FAILURE	0	EXECUTE_FAILURE, reasonCode

In the code, the above code is used for simplifying message sending and receiving.

2. Binder database for scheduling

There are two binder databases, one for functions register and one for server list.

```
std::set<int> server_list;

struct argType{

        char name[BUFFER_SIZE];
        int argTypes[BUFFER_SIZE];
        std::queue<char *> q;
}argTypeDB[20];
```

The above structure is defined to store function name, argument types and server information. For each function, there is a name, an argType, and a queue. The queue stores server address and port number that are registered for this function. Binder uses round robin to schedule the function execution request from clients. When a function execution request is asked, binder first loops through the argTypeDB array to check if the function is registered or not. If so, server information in the front of queue will be popped out and pushed into the end of the queue. Also this information will be send back to the client for execution.

Another database is the server_list set shown above. This one is used to record registered server information. When a rpcTerminate request is received, binder will iterate through the set and send termination request to all the servers.

3. Function overloading

The register and function execution will check both function name and function argTypes. If both of them match, the function is considered to be the same one. Otherwise, it is considered to be different. In this way, same function names with different argTypes would be different functions.

4. Termination procedure

Termination request is sent by privileged clients. When a client chooses to terminate the system, a request will be sent to the binder. The binder recoganizes the termination request, sends another request to all the servers (loop through the server_list set) and waits for all the responses from the server (using a blocking scheme for waiting for servers to terminate). After all the servers are terminated, the binder will terminate.

5. Marshalling/unmarshalling data

Data marshalling follows the rules that: Length, Type, Message

FAILURE_CODE is an integer specifying the reason of failure.

From	To	Message	Content	Executed function
client	binder	LOC_REQUEST	"Length,1,name,argTypes"	argInquiry()
client	binder	TERMINATE	"1,2"	rpcTerminate()
client	server	EXECUTE	"Length,3,name,argTypes,args"	rpcRequest()
binder	client	LOC_SUCCESS	"Length,4,server_identifier,port"	parseLoc()
binder	client	LOC_FAILURE	"3,5,FAILURE_CODE"	parseLoc()
binder	server	REGISTER_SUCCESS	"1,6"	parseRegister()
binder	server	REGISTER_FAILURE	"3,7,FAILURE_CODE"	parseRegister()
binder	server	TERMINATE	"1,2"	rpcTerminate()
server	binder	Terminate ack	"Y"	recvFromBinder()
server	binder	REGISTER	"Length,8,server_identifier,port,name,argTypes"	recvFromBinder()
server	client	EXECUTE_SUCCESS	"Length,9,name,argTypes,args"	recvFromClient()
server	client	EXECUTE_FAILURE	"3,0,FAILURE_CODE"	recvFromClient()

For each separate meaning of message, a comma is used to separate them. Therefore, in the parsing procedure, the comma separator can be used to uniformly parse the message. A string length is at the head of the message telling you how long the message is. Parsing argTypes and args are relatively difficult. ArgTypes are converted into a char array (see the function argTypeToString ()). As to args, we cannot use any string operations. Only memory copy operations are used. Two functions parseArgsToString() and parseStringToArgs() are written for the converting args to a data stream and converting the data stream back to double pointers.

6. Function specification and organization

In the rpc.c file

```
rpcInit(){
        connectBinder();  /*Connect to the binder and prepare registeration"*/
        createClientConn(); /*create a socket for client and listens to it*/
}
rpcRegister(char *name, int *argTypes, skeleton f);
rpcExecute(){
        recvFromClient();
        recvFromBinder();
}
rpcCall(char *name, int *argTypes, void **args){
        argInquiry(char *name, int *argTypes);
        rpcRequest(server_identifier, port, args, name, argTypes);
}
/*Functions for converting an integer to char array
 * several bases are provided, eg: 10, 16, 2, 8*/
char* itoa(int num, char* str, int base);  /*choose base to convert to char array*/

/*functions used for make tcp connections*/
int connectTo(char *address, char *port); /*This function is used to dynamically find a port
number, create a socket and set up connection with the server*/

int socketSend(int socketID, char *message, int length); /*by telling socket id, message to send
and the length of message, this function will send the message to specified id */

/*functions specific for client side*/
int argInquiry(char *name, int *argTypes); /*client send a request to the binder to get the
information of server, who can execute the function*/
int rpcRequest(char *server_identifier, char *port, void **args, char *name, int *argTypes);
/*The request is sent to the server for executing the functions */

/*server specific functions*/
int connectBinder();/*In the rpcInit() function, this is executed to set up connection with the
binder*/
int createClientConn();/*In the rpcInit() function, this is used to set up a server waiting for
clients*/
int recvFromClient(int clientSocketID, fd_set master); /*handle the requests from clients with
the select*/
int recvFromBinder(int binderSock, char *echoBinderBuffer); /*handle requests from binder,
including recving register success/failure, termination*/

/*functions used to parse argTypes and args*/
int equal_arrays(int a1 [], int a2 [], int n, int n1); /*check if two int arrays are exactly the same,
this is used for comparing argTypes array*/
int token(char *input, char sep, char *result); /*use a separator to separate input and save the
beginning part into result, the return value is the length of result + 1*/
void argTypeToString(int *argTypes, char *result, int n); /*convert the argTypes to a char
array*/
void parseArgsToString(int *argTypes, void **args, char *result); /*convert **args to void
array*/
void parseStringToArgs(int *argTypes, void **args, char *str); /*convert a void array to
**args*/
void argTypesCopy(int *argTypesOrigin, int *argTypesTarget); /*copy argTypes to another
one*/
void parseArgTypes(int argTypes, int *result); /*convert the argTypes to an int array of length 3.
result[0] has possible values of 1, 2, 3 which indicating output, input, input and output; result[1]
could be from 1 to 6, indicating the argument types such as char, int, long, etc; result[2] is the
length of the argument. If it is 0, it means it’s a unary. Otherwise it is an array*/
```

int typeLen(int type); /*return the length of type using sizeof()*/

In binder.cpp file

int parseRegister(char *echoString); /*This function is called when there is a function register request from the server. It will store the server information into the relevant argTypeDB[] array. If the function has already been registered, only server information will be stored into the queue; if it's not, a new argTypeDB[] will be created to store the new function information and server information. In the end, if the register is successful, server information will be pushed into a set server_list. */
int parseLoc(char *echoString, int sock, fd_set master); /*This function is called when there is a LOC_REQUEST from the client. Binder will check the argTypeDB[] to see if the function has been registered or not. If existed, it will pop out the front server information in the queue, push this information to the end of the queue and send the information to the client. This is where the round robin scheduling happens.*/
int rpcTerminate(void); /*When the binder receives the termination request, it will forward this request to the servers by iterating through the set server_list */

7. Server duplicate register

```
struct sk{
        skeleton f;
        int argTypes[BUFFER_SIZE];
        char name[BUFFER_SIZE];
} registerDB[20];
```

On server side, the above structure as a global variable is defined and used to store registered functions. When a new register is initiated, the server will loop through the array first to check if this function has been checked. If it is, a warning will be returned. So no duplicate register is guaranteed by the server itself.

Also it is used for checking if it contains the function that a client is asking for.

8 Error code

location	function	error code	specification
server	rpcInit()	-1	Unable to fetch binder location info
server	rpcInit()	-2	Cannot get local address information
server	rpcInit()	-3	Cannot connect to binder
server	rpcInit()	-4	Cannot create a socket for clients
server	rpcInit()	-5	Bind error
server	rpcInit()	-6	listen fails
server	rpcRegister()	-7	send to binder fails
server	rpcRegister()	1	duplicate register warning
server	rpcRegister()	-8	receive from binder fails
server	rpcRegister()	-9	binder fails to register the function
server	rpcExecute()	-10	select fails
server	rpcExecute()	-17	server recieving from clients fails
server	rpcExecute()	-18	server sending to clients fails
client	rpcCall()	-11	Unable to connect to binder
client	rpcCall	-12	Receive from binder fails

client	rpcCall	-13	Connect to server fails
client	rpcCall	-14	Recv from server fails
client	rpcCall	-15	Fuction execution fails(eg. invalid input)
client	rpcCall	-16	The function does not exist on the server
binder	main	-1	socket creation fails
binder	main	-2	bind fails
binder	main	-3	listen fails
binder	main	-4	select fails
binder	main		send() and recv() fails are not considered to be fatal errors, only errors are printed when the operations fail

9. Binder authentication

In this system, only one binder is used. So the authentication is really simple. When server receiving requesting, it simply check to see if the data is from binder socket created on the server machine. Only in this condition, the rpcTermination request can be executed.

Group members:

20499890	King Yiu Tam	ky4tam@uwaterloo.ca
20459717	Jiandi Yao	j25yao@uwaterloo.ca