**CPSC4780 - Course Project**

# Location-Transparent RPC Facility

**Due: to be discussed in class**

**Project goal**

In this project, you are required to create a facility similar to the Sun RPC facility. Though it contains a subset of the functionalities Sun RPC facility provides, it extends it by including *location transparency* for clients. A client can call for a service without knowing where it is.

**Project details**

(1) *myportmapper*

On both the server and client host, there is always one process running, which provides the similar functionalities Sun *portmapper* provides. Let us call it *myportmapper,* in contrast to Sun *portmapper*. You can create it using Sun sockets and it listens to a known port number 10111. It does the following for a client and a server.

A server registers its service through its local *myportmapper*, which then broadcasts this service to other *myportmappers* running on the other hosts in the same distributed system. For the local *myportmapper* on the server host, this service is registered as local while on the other hosts the service is registered as remote (the relevant information required then is the IP address and port number of the service). *myportmapper* will find the next available port number for the service. The corresponding function for registering a service is *my_svc_register(…).*

A client can use a service name to look up for a particular service. It contacts its local *myportmapper* for the service name. If the service does exist, no matter whether locally or remotely, the client gets the service information from the *myportmapper* and start using it through the client stub. The function doing this is *my_clnt_create(…).*

(2) *myrpcgen*

You need to create a RPC generator similar to Sun *rpcgen*, which contains the following functionalities. Let us call it *myrpcgen.*

In the interface protocol file, it can accept the following directives.

`#define constantname constantnumber` (these lines are copied directly to the *.h file)

`program programname` (after this keyword, myrpcgen expects a "{" )

Then a user can define functions. There are only three data types that *myrpcgen* can handle, `int`, `float`, and `char`, to make our job easier. A function can only take one parameter and return one parameter. It should have the following format.

```
        type func(type) = 1;
```

After the remote functions are defined, your *myrpcgen* expects "} = contant".

Note the difference between Sun *rpcgen* and *myrpcgen*. *myrpcgen* does not accept multiple versions of a service, i.e., each service only has one version. As an example, the following is a valid interface definition.

```
#define TWO 2
#define Pi 3.1415

program myprogram {
        int add(int) = 1;
        int sub(int) = 2;
} = 55555;
```

*myrpcgen* will process this interface file and generate the corresponding **.h*, *server stub* and *client stub*.

(3) xdr functions

The following xdr functions are required to be implemented.

```
my_xdr_int(…)          // corresponding to xdr_int(…)
my_xdr_float(…)        // corresponding to xdr_float(…)
my_xdr_char(…)         // corresponding to xdr_char(…)
```

Since we do not want to design and implement the XDR protocols, in our implement the internal XDR format will be just an array.

Note that each of these functions has two modes of operations, *downloading* and *uploading*. When the client sends data to the server, the data will be uploaded into the universal XDR format (in our project, this is an array internally). When the server reads data from the client, the data will be downloaded from the XDR format into the local format. When the returned data is going from the server to the client, we use the same procedure.

(4) Server stub

The following functions need to be provided by your facility for server stub.

```
my_svcudp_create(…)    // corresponding to svcudp_create(…)
my_svctcp_create(…)    // corresponding to svctcp_create(…)
my_svc_register(…)     // corresponding to svc_register(…) but there is no server version
                       // information
```

| | |
|---|---|
| my_svc_run() | // corresponding to svc_run() |
| my_svc_getargs(…) | // corresponding to svc_getargs(…) |
| my_svc_sendreply(…) | // corresponding to svc_sendreply(…) |

Of course, you can implement other auxiliary functions as well for server stub. But the above ones must be implemented.

(5) Client stub

The following functions need to be provided by your facility for client stub.

| | |
|---|---|
| my_clnt_create(…) | // corresponding to clnt_create(…) but there is no server ip address // information. Remember we are implementing location transparent // RPC. |
| my_clnt_call(…) | // corresponding to clnt_call(…) |

Of course, you can implement other auxiliary functions as well for client stub. But the above ones must be implemented.

**Project assumptions**

It would be unrealistic to design and implement the whole RPC facility in a one-semester course. The following assumptions are made to make this project.

(1) There is no error in user's interface file *.x. Your program does not need to check the grammatical errors in the interface file.
(2) Since we do not allow users to define their own new data types, there is no need to generate the file *_xdr.c.
(3) You can define your own protocols between client/server stubs and *my_portmapper*. You need to describe them in detail in your submission.
(4) There are only *four* hosts running in your distributed system.

**Project submission**

(1) Design and implement a solution to the above project using C/C++.
(2) For your project, write a small tutorial (hardcopy) as how to compile and use it. Your marker will check it based on your tutorial.
(3) Write a summary (hardcopy and at most five pages) of your design and implementation, such as the protocol, e.g., data format you have designed, what difficulties you have encountered and how you have handled them, special data structures you have used, etc.
(4) Submit your project (all the source code files, make file, etc.) in a CD such that the marker can check it. Also contained in the CD should be the sample interface files you have used.

**Project hints**

(1) Design and implement *my_portmapper* first, including the protocols to be used by the client/server stubs in order to communicate with it.
(2) Design and implement the client and server stubs next.