Project III: Multi-XML-RPC

Goal

The project will provide you experience with XML-RPC project and add the MultiRPC idea to it.

The Project Requirement

The project consists of three steps

Grab the XML-RPC package, make it run properly on your environment.

Figure out how XML-RPC handles requests. Apply the MultiRPC idea on it.

Try to improve the previous step with a more sophisticated solution.

Step One

First, go to the XML-RPC for C and C++ website. You can learn some basic ideas about how it works. If you are already familiar with it, go directly to the downloading page. Grab the stable version 1.03.11. Extract it in your local Linux environment. Have a look on the README file and your will find out how to make it work. (In case they update the version before you download it, make a note in your submission about the version you use)

The package requires some HTTP client side library. I think the CoC machines all have curl installed. So we will default choose curl as the client side. After your run make, there should be one special header file 'transport_config.h' generated. It should look like

```
/* This file was generated by a make rule */
#define MUST_BUILD_WININET_CLIENT 0
#define MUST_BUILD_CURL_CLIENT 1
#define MUST_BUILD_LIBWWW_CLIENT 0
static const char * const XMLRPC_DEFAULT_TRANSPORT = "curl";
```

If the MUST_BUILD_CURL_CLIENT is not defined as 1, your system has no curl installed. Install the curl yourself or contact me. If the default transport is not curl, change it so. After you compiled the package, you should be able to test it using the applications in the examples folder.

Step Two

Now we will play with the MultiRPC idea. MultiRPC is an extension to RPC that provides a parallel RPC capability for sending a single request to multiple servers and awaiting their individual responses. In this step, you need trace the call stack of the client request call xmlrpc_client_call_asynch. Follow the call stack, you will reach xmlrpc_client_call_server_asynch_params, then sendRequest, then clientP->clientTransportOps.send_request which should be located in inxmlrpc_curl_transport.c if curl is used. A little further, you will notice that a thread is created and the request is handled there.

To apply the MultiRPC idea, you will define a new asynchronous call that accepts multiple server URLs. Add all necessary code to support the multiple Server URLs. And you will create multiple threads instead of one, each sends a request to one server. Only the first complete one can call the callback function (complete). Also pay attention to the cleanup work of your multiple threads.

Please note that failure on one server should not bring down the RPC call. the RPC should success if at least one server response successfully. Your program should generate some kind of logs to show how the requests are sent out to multiple servers and which server response is picked first. You can use the message output already inside of the XML-RPC or you can write your own log file. I only need to see something to demonstrate the correctness of your mechanism. Please make it clear in your README file that how can I check it.

Step Three

The above method only works on asynchronous calls. The synchronous call xmlrpc_client_call will not create a separated thread thus we cannot use the above method to extend it. The call will finally reaches curl_easy_perform. Noticed that curl has function support for process multiple handles: curl_multi_perform. So in this step, you will investigate whether it is possible to send the request to multiple servers and wait for the first response without creating multiple threads for each request. To make your work easier, you don't need to create a new synchronous call for MultiRPC. you can still use the asynchronous one you have done in step two. Create only one thread and do all the work in it.

If you find that curl has enough support to do it this way, implement the MultiRPC support using a single thread. If you find out that curl is not strong enough to do it, write what you find in your report file, explain why it can not be done and what should be added to the curl library to make it possible.