Distributed Password Cracker using ONC RPC

Ankita Girish Wagh
UIN : 621003584
Sambit Mishra
UIN : 720002013

April 2, 2013

1 Design:

In order to implement the Distributed Password Cracker using ONC RPC , on the given code we changed the LSPMessage and the socket connections from the previous assignment. In order to remove the LSPMessage and still continue to transfer messages we created a structure of our own called "message'. The message structure was defined inside the **"server_stub.x"** stub file. The stub file was compiled using rpcgen compiler. It created 4 files i.e. server_stub.h , server_stub_clnt.c, server_stub_svc.c , server_stub_xdr.c. These files contained the RPC functions along with the signature and declarations.

2 Architecture

Our architecture is a **"Polling Architecture"**. We have created 2 RPC methods i.e. int * server_recv_mssg_1_svc(message *, struct svc_req *) and message * server_send_mssg_1_svc(uint32_t *, struct svc_req *). The server_recv_mssg_1_svc is invoked from the client side when it has to transfer any packet to the server. This RPC call fills in the server_input queue. The server read thread picks up the message and handles it accordingly. The client keeps on polling the output queue of the server to get its packet. In order to carry out the polling the client invokes the server_send_message_1_svc() function. It inspects the queue and picks up its own packet. We have created a map to store the connection id and address[port number] of the client. This helps the client in indentifying its own packet.

3 RPC Functions

int * server_recv_mssg_1_svc(message *, struct svc_req *) function fills in the server input queue. message * server_send_mssg_1_svc(uint32_t *, struct svc_req *) function returns the message from the server output queue to the clients.

The stub code contains the signature of these functions. The rpcgen compiler generates the code other files which are needed for the client to invoke the functions in the server side.

There has been no change in the upper layers. All the change was carried out in the network handler layer. We have removed all socket "recvfrom" and "sendto()" calls with the respective RPC methods. The major change in architecture as compared to the previous assignment is that the **server** is dormant. It picks up message and processes it. Once the message has been processed , the server fills in the output queue. As it is a polling architecture the clients keep on polling and pick up their corresponding message by using the connection id and required sequence number.

We have created 2 new queues namely "global_queue" and "global_output_list" to store the incoming and outgoing messages to the clients respectively. The global_output_list stores a structure of rpc_message. This stores both the message and the sockaddr_in address of the clients.

4 Execution

In order to execute our code carry on the following steps. We have already generated the stub files using rpcgen compiler from our "server_stub.x" files.

Execution Steps:
make clean
make -f Makefile

This will generate three binaries i.e. server , request , worker.

Start the Server :sudo ./server 127.0.0.1:9090
Note : The server needs to be started with sudo access rights

Start the Client : ./request 127.0.0.1:9090 fbec17cb2fcbbd1c659b252230b48826fc563788 4

Start the Worker : ./worker 127.0.0.1:9090

Result : Found : rain