Lab: Remote Procedure Call – RPC

Remote procedure call (RPC) is the invocation of a function that resides in a remote process or server, and the invocation has the same appearance as a local function call. The need for RPC is motivated by performance considerations which prompt the spreading of functions and functionality over multiple hosts, as well as by the necessity of remote function invocation in cases where the information and data are only available on remote servers.

There are a number of different ways to do RPC. In Python, you can use the XML-RPC or JSON-RPC libraries. Each of these libraries supports a specific data structure that is dictionary-like. This proves to be a limitation. We will use the RPyC library which is a more powerful RPC library that is not limited to a specific data structure.

Install the RPyC library

Open command prompt and type: `pip install rpyc`

An example

We show how to use RPyC to do RPC using a simple example. A server has one function that is open to the outside for remote invocation (i.e., RPC). This function is called `count_lines()` and takes two arguments: a file object and a function. Recall that in Python, functions can be passed into and returned from a function, and these functions can be called as usual. In the `count_lines()` function, it iterates through the file (the file object passed in) line by line, and does a **callback** of the function that's passed in for each line. **Note that this function being called lives in the remote client.** It returns the total number of lines in the file at the end.

The client establishes a connection to the server first and creates a `proxy` which stands for the server process. Then it can proceed to remotely call the function residing in the server, as if it's a local function.

The server program is in `rpc_server.py`:

```
import rpyc

class RemoteService(rpyc.Service):
    def exposed_count_lines(self, fo, function):
        print('Client has remotely called exposed_count_lines()')
        for n, line in enumerate(fo):
            function(line)
        return n+1

if __name__ == '__main__':
    from rpyc.utils.server import ThreadedServer
    thr = ThreadedServer(RemoteService, port = 12358)
    thr.start()
```

It is **important** to point out that a function can only be remotely called if the server names it with the prefix `exposed_`.

The client program is in `rpc_client.py`:

```
import rpyc

def echo(string):
    print('echo:', repr(string))

if __name__ == '__main__':
    config = {'allow_public_attrs': True}
    proxy = rpyc.connect('localhost', 12358, config=config)
    fo = open('afile.txt')
    numlines = proxy.root.count_lines(fo, echo)
    print('Number of lines in file = ', numlines)
```

The function that the client passed to the remote function invocation is called `echo()` and all it does is echo the line that is passed back from the server. Notice that the server actually is doing a remote callback of this function residing in the client process.

You'll notice that there is some configuration set when the client creates a proxy by connecting to the server. This configuration allows you to set up more restricted access to RPC of server functions for security reasons. Here, we simply give complete control to the server and do not set any restrictions on the permissions, by setting `allow_public_attrs` to be true.

Run the programs

The file passed by the client in the remote function call is named `afile.txt` and it's provided, it is just a copy of the client program.

Run the RPC server first: `python rpc_server.py`

Then run the client: `python rpc_client.py`

Task

You will implement a server that provides an election service. It provides two functions that can be called remotely by clients: (1) *vote*. It has two parameters through which the client supplies the name of a candidate (a string) and the "voter's number" (an integer used to ensure each user votes once only). The voter's numbers are allocated sparsely from the range of integers to make them hard to guess. It has a third parameter that is a callback function from the client. (2) *result*. This method has two parameters through which the server supplies the client with the name of a candidate and the number of votes for that candidate.

The third parameter the client passes to the `vote` function is a callback function, e.g., called message(). The server can call this function to pass messages to the client. When a user votes, the server passes a string to this function that confirms the vote the user just cast. In the client-side message() function, the string is printed. If a user tries to vote more than once, the server should discard the duplicate vote and call the message function to send a message informing the user what happened.

The Election service must record a vote whenever a user thinks they have cast a vote. Each user can only vote once. A duplicate user vote should be detected by the server and discarded, and the server should use the client callback function to send a message stating what happened. The records should remain consistent when it is accessed concurrently by multiple clients.

Answer the following questions:

1. Using RPyC, what distinguishes syntax-wise between functions that can be remotely invoked and those that cannot?

2. Why does doing the above make it more secure for the server?

3. How would you modify the implementation to ensure that the votes are recorded even when the server process crashes (and later restarts)?

To submit

Your programs for the task, screen shot of your programs running, and answers to the questions.