Assignment explanation & guideline

> Name: Jinglong Li
>
> NYU ID: N13903873

1. Explanation of the code

In this assignment, we take use of the remote procedure call.
Rpcgen can help us automatically generate RPC client and server stubs in C, by taking an .x(interface definition) file as an input.
There will generate 7 files, in which we only need to modify the client.c and server.c file to implement the functionality.

In client.c file,

Using the following function to set up an connection.

```
CLIENT *clnt_create(const char *host,
                    const u_long prognum,
                    const u_long versnum,
                    const char *nettype);
```

In my homework , I take use of the UDP protocol.

```
clnt = clnt_create (host, STRING, VERSION, "udp");
if (clnt == NULL) {
    clnt_pcreateerror (host);
    exit (1);
}
```

we take use of the function generated in clnt.c to communicate with the server,

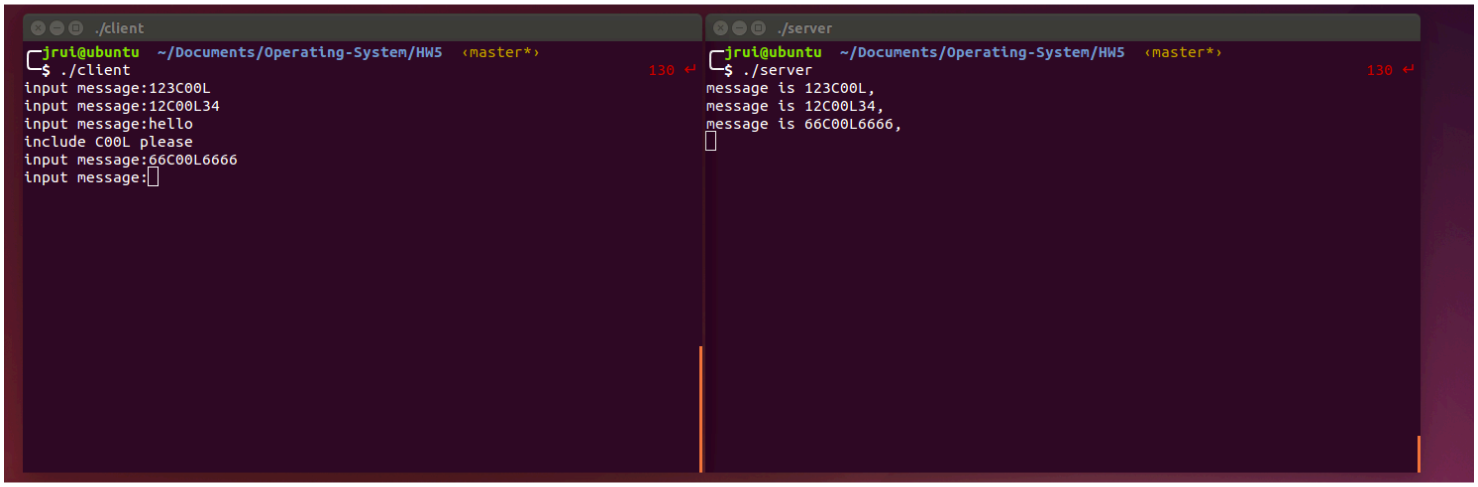
```c
char *comp_str = strstr(message,"C00L"); //compare if input message contains "C00L
if(comp_str!=NULL){
    result_1 = tst_1(&message, clnt); // tst_1 is the function defined in string.c
} else {
    printf("include C00L please\n");
}
if (result_1 == (char **) NULL) {
    clnt_perror (clnt, "call failed");
}
```

In the server file, it is already generated the main frame of the program, what we need to do is add our functional module into it.

So:

1. `$rpcgen string.x` will generate `string.h` 、 `string_svc.c` 、 `string_clnt.c`
2. `gcc string_client.c string_clnt.c -o client` will generate `client`
3. `gcc string_server.c string_svc.c -o server` will generate `server`

The result is like this:



```
./client
┌jrui@ubuntu  ~/Documents/Operating-System/HW5  ‹master*›
└$ ./client                                              130 ↵
input message:123C00L
input message:12C00L34
input message:hello
include C00L please
input message:66C00L6666
input message:
```

```
./server
┌jrui@ubuntu  ~/Documents/Operating-System/HW5  ‹master*›
└$ ./server                                              130 ↵
message is 123C00L,
message is 12C00L34,
message is 66C00L6666,
```

2. Guideline

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

Steps in RPC:

1.Client Program makes a remote procedure call
2.Stub calls the RPC runtime library routines to establish a connection to the service daemon
3.Client sends in the parameters and the informaiton about the remote procedure 4.Server daemon dispatches service procedure
5.Remote procedure is executed on the remote machine
6.The answer is assembled and returned to the client
7.Client proceeds to the next instruction

To develop an RPC application the following steps are needed:
1.Specify the protocol for client server communication 2.Develop the server program
3.Develop the client program

Defining the Protocol:
The easiest way to define and generate the protocol is to use a protocol compiler such as rpcgen.For the protocol you must identify the name of the service procedures, and data types of parameters and return arguments. The protocol compiler reads a definitio and automatically generates client and server stubs.
for example:

```
program MESSAGEPROG {
    version PRINTMESSAGEVERSION {
        int PRINTMESSAGE(string) = 1;
    } = 1;
} = 0x20000001;
```

Program name and procedure names are declared in all CAPS. Not required, but is a standard convention.

- Remote Procedures are declared in a remote program (messageprog)
- Remote Procedures need to be specified in the protocol file.
- In this case a single procedure printmessage is declared.
- The argument to the procedures and their types, are declared here.
- The procedures are numbered (in this case 1).
- Programs have associated version numbers, (in this case 1).
- Change the version number when new functionality is added.
- Multiple versions of the same program may exist.
- Programs also have program numbers (in this case 0x2000000).

If we run rpcgen -a -C rpcprog.x in the console, it will automatically generate several files we need to implement the RPC communication.

Develop the server program:
There will be an file called rpcprog_server.c, we need to modify this file to set the remote procedure as a server in assignment3, there is only one function and you can add your code in it to achieve your goals.

Develop the client program:
In our folder, there will be a file called rpcprog*client.c, it use the function in the server program to call the remote procedure.*
But at first we need to build a client to call the function on the server.
*Here is the clnt*create function.
client = clnt_create(server, MESSAGEPROG, PRINTMESSAGEVERSION, "udp") “client” is the second parameter of the remote procedure. The first parameter of the remote procedure call is what we want to send the server.