

# Sun(ONC) RPC

《操作系统课程设计》实验指导

同济大学计算机系 邓蓉

2024/3/31

参考文献: <https://people.cs.rutgers.edu/~pxk/417/notes/rpc/step4.html>

本教程是 Sun RPC 编程基础。我们写一个执行加法操作的 RPC 服务器。客户端从命令行得到 2 个整数，送给 RPC 服务器，后者运算，送回加法操作的结果，供客户端显示。

## 第一步：创建 IDL

我们定义符合 Sun IDL (Interface Definition Language) 格式的接口文件 (后缀名是.x)。文件内容是一个 program，也就是一个 RPC 服务。包括若干 version，这个 RPC 服务的不同版本。各个版本独立对外服务，包括需要使用的所有数据结构和过程。

传统上 RPC 只接受一个参数，现在不是了。但我们还是遵守最初的约束条件，将所有参数写进一个数据结构。

例：加法服务器的 IDL 文件，add.x。

```
struct intpair {
    int a;
    int b;
};

program ADD_PROG {
    version ADD_VERS {
        int ADD(intpair) = 1;
    } = 1;
} = 0x23451111;
```

加法服务器的程序号是 0x23451111；只有一个版本，版本号是 1；这个版本只有一个函数 add，函数 ID 是 1。

编译它（执行存根生成器）

```
rpcgen -C add.x
```

-C 标识告诉 rpcgen 生成符合 ANSI C 标准的 C 代码。看下，它生成了 4 个文件。详情见附录一。客户端、服务器都要使用的头文件 add.h，服务器主程序 add\_svc.c，客户端存根 add\_clnt.c 和 参数打包程序 add\_xdr.c。

## 第二步：生成样例 客户端、服务器 代码

`rpcgen` 可以生成客户端、服务器所需的全部代码，包括所有的模板程序和 `Makefile`。这样做：

```
rpcgen -a -C add.x
```

重命名 `Makefile.add`：

```
mv Makefile.add Makefile
```

修改 `Makefile` 文件，将编译器改成 `gcc`。我们在 `Makefile` 里加一行，这一行要放在 `CFLAGS=` line 之前。

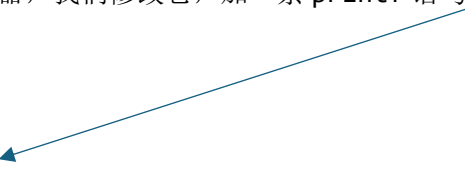
```
CC=gcc
```

## 第三步：测试 client 和 server

模板程序 `add_client.c` 是客户端主程序，先别动，过会再来修改它。它有一个命令行参数，是服务器的名字，创建一个连接服务器进程的 RPC 句柄，调用 `add_1` 函数。

模板程序 `add_server.c` 是服务器，我们修改它，加一条 `printf` 语句在这里，换掉注释：

```
.....  
/*  
 * insert server code here  
 */  
.....
```



要修一下 `Makefile`，定义符号 `RPC_SVC_FG`，这会让我们加法服务器运行在前台。调试的时候很方便。把

```
CFLAGS += -g
```

改成

```
CFLAGS += -g -DRPC_SVC_FG
```

```
RPCGENFLAGS =
```

改成

```
RPCGENFLAGS = -C
```

完成后，`make`。输出大致这样：

```
gcc -g -DRPC_SVC_FG -c -o add_clnt.o add_clnt.c  
gcc -g -DRPC_SVC_FG -c -o add_client.o add_client.c  
gcc -g -DRPC_SVC_FG -c -o add_xdr.o add_xdr.c  
gcc -g -DRPC_SVC_FG -o add_client add_clnt.o add_client.o add_xdr.o -lnsl  
gcc -g -DRPC_SVC_FG -c -o add_svc.o add_svc.c  
gcc -g -DRPC_SVC_FG -c -o add_server.o add_server.c  
gcc -g -DRPC_SVC_FG -o add_server add_svc.o add_server.o add_xdr.o -lnsl
```

第一次玩 RPC，要安装 `rpcbind`。

```
sudo install rpcbind
```

启动加法服务器。

```
./add_server
```

再开一个终端，启动客户端。

```
./add_client localhost
```

好的，我们空的服务器启动了。看，它有输出了“add function called”。

## 第四步：让服务器工作

做一些有意义的事情。

编辑 `add_client.c`，在调用 `add_1` 函数之前，准备参数。

编辑 `add_server.c`，接收客户送来的参数，加一加，送回计算结果。

`make`，重新编译。

先启服务器 `./add_server`

再启客户端 `./add_client localhost 10 20`

一切正常，任务完成。

以下，是我写的代码样例：阴影处和黄色下划线是加入的代码。

```
C add_client.c X
C: > Users > Pretty-Girl > Desktop > add_client > C add_client.c
1  #include "add.h"
2
3  void
4  add_prog_1(char *host,int x,int y)
5  {
6      CLIENT *clnt;
7      int *result_1;
8      intpair add_1_arg;
9
10
11     #ifndef DEBUG
12         clnt = clnt_create (host, ADD_PROG, ADD_VERS, "udp");
13         if (clnt == NULL) {
14             clnt_pcreateerror (host);
15             exit (1);
16         }
17     #endif /* DEBUG */
18     add_1_arg.a = x;
19     add_1_arg.b = y;
20     result_1 = add_1(&add_1_arg, clnt);
21     if (result_1 == (int *) NULL) {
22         clnt_perror (clnt, "call failed");
23     }
24     printf("result = %d\n", *result_1);
25     #ifndef DEBUG
26         clnt_destroy (clnt);
27     #endif /* DEBUG */
28 }
29
30
31 int
32 main (int argc, char *argv[])
33 {
34     char *host;
35
36     if (argc < 2) {
37         printf ("usage: %s server_host\n", argv[0]);
38         exit (1);
39     }
40     host = argv[1];
41     add_prog_1 (host,atoi(argv[2]),atoi(argv[3]));
42     exit (0);
43 }
```

```
C add_client.c  C add_server.c X
C: > Users > Pretty-Girl > Desktop > add_client > C add_server.c
1  /*
2   * This is sample code generated by rpcgen.
3   * These are only templates and you can use them
4   * as a guideline for developing your own functions.
5   */
6
7  #include "add.h"
8
9  int *
10 add_1_svc(intpair *argp, struct svc_req *rqstp)
11 {
12     static int result;
13
14     // printf("add function called\n");
15
16     result = argp->a + argp->b;
17     printf("returning: %d\n", result);
18
19     return &result;
20 }
21
```

# 附录一

## add.h

client 程序和 server 程序需要使用的头文件。有我们定义的结构 `intpair`，客户端存根 client stub 接口(`add_1`) 和 我们要改写的服务侧函数(`add_1_svc`)的接口。此外，符号 `ADD_PROG` (`0x23451111`), `ADD_VERS` (`1`)和 `ADD`(`1`)分别是程序号、版本号和函数号。

This is the header file that we'll include in both our client and server code. It defines the structure we defined (`intpair`) and *typedefs* it to a type of the same name. It also defines the symbols `ADD_PROG` (`0x23451111`, our program number) and `ADD_VERS` (`1`, our version number). Then it defines the client stub interface (`add_1`) and the interface for the server-side function that we'll have to write (`add_1_svc`). In the past (pre ANSI-C), the client stub and server side functions had the same name but since ANSI C was strict with parameters matching their declarations, this was changed since the parameters are slightly different. As we'll soon see, the client stub accepts an extra parameter representing a handle to the remote server. The server function gets an extra parameter containing information about who is making the connection.

## add\_svc.c

这是服务器程序，有 `main` 函数。它注册 `RPC` 服务，之后对外提供加法服务。  
函数 `add_prog_1` 是程序的监听器，主要是 `switch` 分支，我们的加法函数（服务器 `add_1_svc`）在里面。监听器运行时，根据函数号进不同的分支，设置函数指针 `local`，我们的 `add` 函数，`add_1_svc`；对输入参数进行反序列化处理之后，调用本地函数 `add_1_svc`。

This is the server program. If you look at the code, you'll see that it implements the *main* procedure which registers the service and, if the symbol `RPC_SVC_FG` is not defined, forks a process to cause the service to run in the background; the parent exits.

The program also implements the listener for the program. This is the function named `add_prog_1` (the `_1` is used to distinguish the version number. The function contains a switch statement for all the remote procedures supported by this program and this version. In addition to the *null* procedure (which is always supported), the only entry in the switch statement is `ADD`, for our `add` function. This sets a function pointer (`local`) to server function, `add_1_svc`. Later in the procedure, the function is invoked with the unmarshaled parameter and the requestor's information.

## add\_clnt.c

定义了客户端 stub 函数 `add_1`。这个函数 序列化参数，执行 `RPC`，返回结果。

This is client stub function that implements the `add_1` function. It marshals the parameter, calls the remote procedure, and returns the result.

## add\_xdr.c

对 `intpair` 结构中的参数进行序列化。

The `_xdr.c` file is not always generated; it depends on the parameters used for remote procedures. This file contains code to marshal parameters for the `intpair` structure. It uses XDR (eXternal Data Representation) libraries to convert the two integers into a standard form.