**软件设计说明和描述文档**

**杨顺云 15060107**

1. 软件项目结构：

RPC：项目文件夹

--bin：可执行程序文件夹

--client：客户端可执行程序

--server：服务端可执行程序

--include：头文件文件夹

--requestservices.h：客户端请求服务的相关方法申明

--services.h：服务端提供的服务原型申明

--preservices.h：服务端对原服务统一封装的一致性接口申明

--tools.h：服务端一些通用辅助方法的申明（如hash，转型，XDR…）

--src：源程序文件夹

--client：客户端源程序

--requestservices.c：客户端请求服务的相关方法实现

--tcp\_client.c：客户端主程序（请求服务，获取远程结果…）

--server：服务端源程序

--services.c：服务端提供的服务原型实现

--preservices.c：服务端对原服务统一封装的一致性接口实现

--tools.c：服务端一些通用辅助方法的实现

--tcp\_server.c：服务端主程序（注册服务，hash查找，并发反馈…）

--Makefile：项目编译生成可执行程序的makefile文件

1. 函数说明：

（一）客户端部分：

（1）requestservices.c：

1、void startRequest(char\* funcName, char\* paraList, char \*ip, char \*result)：

对服务名，服务参数列表字符串进行XDR外部编码后向远程主机请求服务，并将得到的反馈结果存储于result字符串之中。

2、void requestService(char\* buf ,char \*ip, char\* result)

被startRequest函数所调用，由XDR编码后的字符串利用socket编程执行具体的远程请求操作，将得到的反馈结果存储于result字符串之中。

（2）tcp\_client.c:

1、void main(int argc ,char \*argv[])

客户端主函数，根据目标IP执行远程服务请求。业务流程：循环输入请求的服务名和参数列表字符串（逗号分隔各参数），然后打印得到的反馈结果（可能是正确的结果，可能没有该服务，也可能服务参数错误）。直到输入的服务名为exit时结束客户端程序。

（二）服务端部分：

（1）services.c：

1、char\* sayHello(char myName[], int myAge)

sayHello服务的具体实现。

2、int sum( int item1, int item2)

sum服务的具体实现，两个数字作加法。

3、char\* sayGoodbye(char myName[])

sayGoodbye服务的具体实现，类似sayHello。

注：目前暂时测试提供以上三种服务。

（2）preservices.c：

1、char\* pre\_sayHello(char \*str)

解码远程访问字符串，拆分其中的参数列表，对参数进行正确性检查，判断

其是否满足sayHello服务的参数要求，检测通过则调用具体的sayHello服务，返回执行结果；检测不通过，返回相应错误提示。

2、char\* pre\_sum(char \*str)

解码远程访问字符串，拆分其中的参数列表，对参数进行正确性检查，判断

其是否满足sum服务的参数要求，检测通过则调用具体的sum服务，返回执行结果；检测不通过，返回相应错误提示。

3、char\* pre\_sayGoodbye(char \*str)

解码远程访问字符串，拆分其中的参数列表，对参数进行正确性检查，判断

其是否满足sayGoodbye服务的参数要求，检测通过则调用具体的sayGoodbye服务，返回执行结果；检测不通过，返回相应错误提示。

（3）tools.c：

1、int HashString(char \*str ,int numBins)

字符串Hash函数，numBins用于取模，一般是服务端可提供服务的上限。Hash的功能用于快速地由服务名定位到具体的服务，避免服务过多时进行循环查找造成的巨大开销。

2、int findEmptyPos(struct func \*funcs,char \*name, int numBins)

由服务名name查找一个hash表funcs中的可用位置，为注册服务作准备。简单地采用开放定址法解决冲突。（当然现在测试只提供三个服务，基本没有冲突的可能）。

3、int findFunction(struct func \*funcs,char \*name,int numBins)

由服务名name查找一个hash表funcs中的具体服务，因为可能是之前发生冲突后添加的，不能确定一定可以一次匹配，还是要用开放定址法，找到函数名一致为止。

4、void decodeFunc(char \*str,char \*funcName,char \*paraList);

解码字符串，得到服务名和参数列表。

5、void spiltStr(char \*str, char params[PARAM\_MAX][PARAM\_LEN] , int \*num)

拆分参数列表，得到每一个参数字符串，以及参数的个数。

6、int isInt(char \*str)

判断字符串能否转换成整型数。

7、int isFloat(char \*str)

判断字符串能否转换成浮点数。

（4）tcp\_server.c：

1、void initServices(struct func \*funcs)

初始化hash表，将每项的flag域置空，表示尚未添加服务。

注：hash表每一项的结构定义如下：  
struct func

{

char name[50]; //函数名

char\* (\*f)(char \*args); //服务的入口地址

int flag; //该服务是否已经注册存在

};

2、void registerServices(struct func \*funcs, char name[] , char\* (\*f)(char \*))

由函数名和开放定址的策略查找hash表中的空位，完成服务的注册。

3、void StartService(struct func \*funcs)

启动监听服务。等待客户端请求，一旦连接过来，用fork()开启子进程完成

具体的服务查询，计算和反馈工作。父进程继续监听。以此完成服务端的并发控制工作。子进程处理的业务流程：解析请求，由服务名进行hash查询，查不到则反馈无服务信息。查到后进入一致性服务接口，判断参数列表的正确性。参数有误，反馈参数错误信息。参数正确则执行具体的服务业务，反馈执行结果。

4、void main()

服务端主程序：初始化hash表，注册服务，启动监听。

1. 两端交互业务流程图：



图1 客户端服务端业务交互流程图

1. 项目使用说明：
2. 实验环境：

本实验采用Linux下的C语言编程，其自带Socket和Xdr函数库，可以直接调用网络通信和XDR外部编码相关接口完成工作。

操作系统：虚拟机下的64位CentOS 6.5

开发环境：文本编辑器、gcc编译器

1. 软件使用：
2. 拷贝项目RPC文件夹到linux系统某目录下。
3. 进入RPC文件夹，执行make all进行项目编译和可执行文件生成（本来bin目录下是有可执行文件的，但这是在我的系统下编译生成的，不一定适用于您的系统，所以可能需要自行再次生成，Makefile脚本已写好，一句命令即可完成工作）。
4. 执行bin/server启动服务监听。
5. 执行bin/client ip连接服务器开始服务请求业务流程（具体过程参见上面的流程图）

注：bin/server和bin/client ip可以分别执行在两台可以互相通信的主机之上，同时因为服务端实现了并发控制，多个客户端可以一起请求服务。

1. 总结：

通过本次实验，我熟悉了面向过程的RPC软件开发，深入地掌握了socket网络编程的实现原理和机制，了解了不同机器之间网络传输数据存在的问题，并通过老师教授的解决方案（XDR）解决了该问题。感觉获益颇多。

同时，实验过程中也遇到了诸多的问题，比如一开始为了方便，我是想在熟悉的windows环境下通过VS之类的IDE便捷地完成任务。但编了一部分，正要用到外部编码技术时突然发现windows下怎么也找不到可用的XDR库。最后不得不放弃windows，改用linux，还是在虚拟机下纯用文本编写，自行管理编译整个项目。好在毕竟只是课程实验，规模不能算特别大（当然也可以完善的很大）。最后还是较为完整地完成了整个工作。感觉对软件项目的规划管理也有了更进一步的理解和经验。

总之，在整个实验中这个不断发现问题，解决问题的过程才是让我获益和感触最多的部分。感谢老师给我们提供这次实践的机会。