



**中国科学院大学**  
University of Chinese Academy of Sciences

## 《高级操作系统》设计报告

报告题目 RPC 模拟设计与实现

小组成员 秦伟、张伟、陈瑶瑶

研究所（院系） 中科院沈阳计算技术研究所

2020 年 10 月 1 日

## 目 录

1 设计任务 .....	1
1.1 设计内容.....	1
1.2 实现环境.....	1
1.3 组内分工.....	1
2 设计原理 .....	2
2.1 RPC 工作原理.....	2
2.1.1 请求过程: .....	2
2.1.2 应答过程 .....	3
2.2 TCP/IP 协议.....	3
2.2.1 TCP/IP 体系结构 .....	3
2.2.2 TCP/IP 工作原理 .....	4
3 设计过程 .....	5
3.1 总流程图.....	5
3.1.1 服务器程序 .....	5
3.1.2 客户端程序 .....	5
3.2 传输数据结构.....	6
4 运行结果 .....	7
4.1 传输成功运行结果.....	7
4.2 客户端未找到服务器运行.....	8
4.3 测试字符串长度超出 1024 .....	8
5 总结与反思 .....	10
附录: .....	11

## 1 设计任务

### 1.1 设计内容

设计程序模拟客户机和服务器，利用 RPC 来调用 `printf` 函数实现客户机与服务器之间的通信。

### 1.2 实现环境

Windows10; Microsoft Visual Studio 2010;

### 1.3 组内分工

如图 1.1:

姓名	学号	工作任务
秦伟	2020Z8013382012	负责小组内成员工作划分；设计实现 RPC，完成代码书写；完成设计报告中的“设计过程”和“运行结果”部分。
张伟	2020Z8013382011	辅助代码书写工作；完成设计报告中“设计原理”部分；完成设计报告的排版工作。
陈瑶瑶	2020Z8013382002	查找 RPC 相关技术文档，对设计的要求完成解释说明工作；完成实验报告“设计内容”和“设计原理部分”。

图 1.1 组内分工

## 2 设计原理

### 2.1 RPC 工作原理

RPC 是一种 C/S 编程模式，比 C/S Socket 更高一层。RPC 是将一个服务的请求和执行在客户和服务端之间进行分布。客户在请求一个远程服务之前，要知道服务的接口名和调用参数，然后它通过网络向服务器发出 RPC 请求。服务器收到请求后，调用本地的服务，然后将结果传回客户。当建立 RPC 服务以后，客户端的调用参数通过底层的 RPC 传输通道(UDP 或 TCP)，并根据传输前所提供的目的地址及 RPC 上层应用程序号转至相应的 RPC 服务程序，且此时的客户端处于等待状态，直至收到应答或 Time Out 超时信号。当服务器端收到请求消息，根据注册 RPC 时告诉 RPC 系统的例程入口地址，执行相应的操作，并将结果返回至客户端。调用流程如图 2.1:

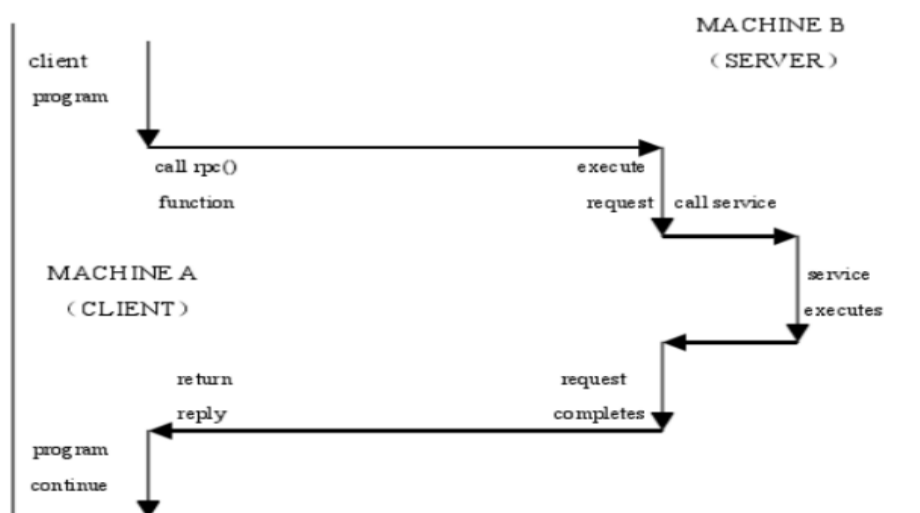


图 2.1 RPC 的一般工作流程

#### 2.1.1 请求过程:

Step1:客户端函数将参数传递到客户端句柄。

Step2:客户端句柄将请求序号、远程方法、参数等信息封装到请求对象中，并完成请求对象序列化形成请求报文，通过网络客户端发送请求报文。

Step3:请求报文通过网络客户端与网络服务端所约定的协议（HTTP、RMI 或自定义）进行通讯。

Step4:网络服务端收到请求报文之后,通过反序列化,从请求对象中解析出远程方法、参数等信息,并根据这些信息找到服务器句柄。

Step5:通过服务器句柄完成服务器函数的本地调用过程。

自此,整个请求流程完成。

### 2.1.2 应答过程

Step1:服务器函数执行的过程将结果返回服务器句柄,返回的结果可能是正常返回,也可能是以抛异常的形式返回。

Step2:服务器句柄根据返回的值与请求序号封装到应答对象中,并完成应答对象的序列化,形成应答报文,通过网络服务端发送应答报文。

Step3:应答报文通过网络服务端与网络客户端所约定的协议(HTTP、RMI 或自定义)进行通讯。

Step4:网络客户端收到应答报文之后,通过反序列化,从应答对象中解析出请求序号所挂钩的客户端句柄。

Step5:客户端句柄将返回数据返回到客户端函数,以返回值或抛异常的形式将信息返回。

自此,整个应答流程完成。

## 2.2 TCP/IP 协议

### 2.2.1 TCP/IP 体系结构

TCP/IP 协议并不仅仅包括 TCP 和 IP 这两个协议,而是泛指 Internet 上的 TCP/IP 协议族。TCP/IP 是一个四层的体系结构,它包含应用层、传输层、网络层和网络接口层。

## 2.2.2 TCP/IP 工作原理

在源主机上应用层将一串字节流传给传输层；传输层将字节流分成 TCP 段，加上 TCP 包头交给 IP 层；IP 层生成一个包，将 TCP 段放入其数据域，并加上源主机和目的主机的 IP 地址后，交给网络接口层，然后再交给数据链路层，数据链路层在其帧的数据部分装上 IP 包，发往目的主机或 IP 路由器处理。

在目的主机处，数据链路层将数据链路层帧头去掉，将 IP 包交给网络接口层再交给 IP 层，IP 层检查 IP 包头，如果包头中的检查和计算出来的不一致，则丢弃该包；如果检查一致，IP 层去掉 IP 头，将 TCP 段交给 TCP 层，TCP 层检查序号来判断是否为正确的 TCP 段；TCP 层检查 TCP 包头，如果不正确就抛弃，若正确就向主机发送确认；目的主机在传输层去掉 TCP 包头，将字节流传给应用程序。工作流程如图 2.2 所示。

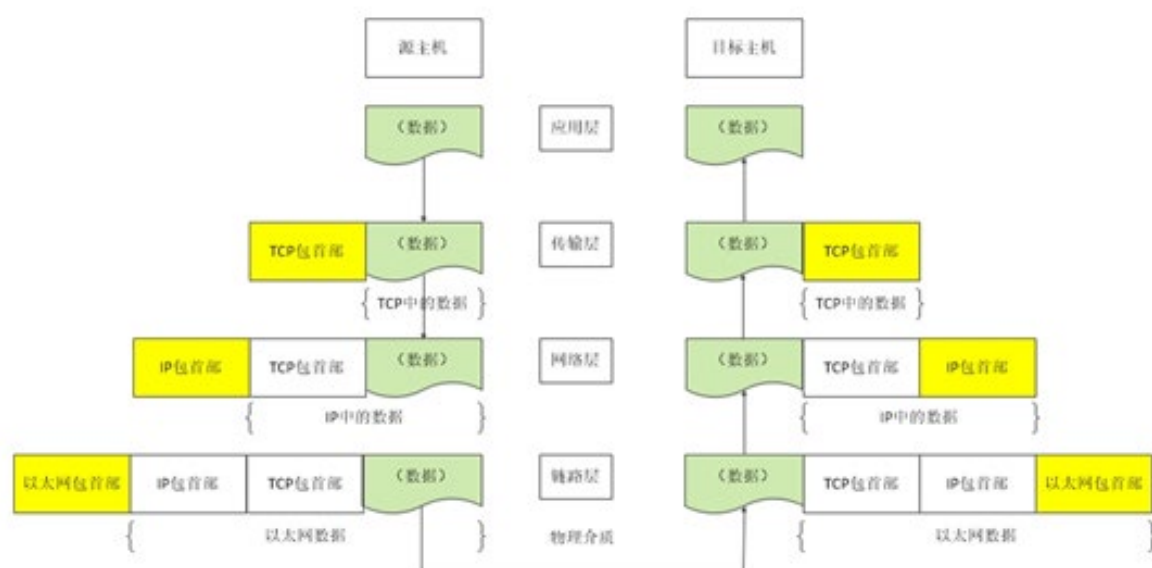


图 2.2 TCP/IP 的工作原理

### 3 设计过程

#### 3.1 总流程图

如图 3.1:

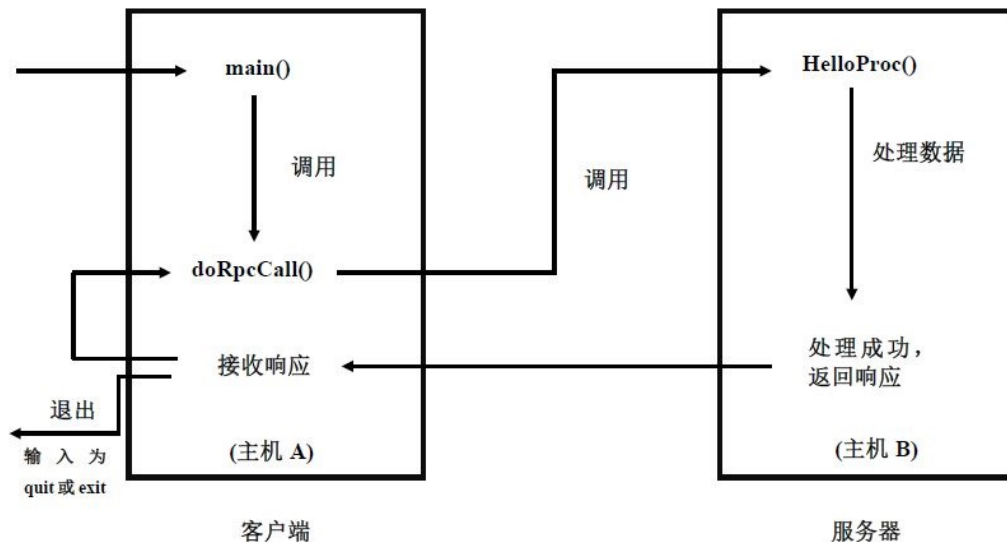


图 3.1 RPC 总体流程图

RPC 是基于客户 / 服务器模式的，因此被划分为客户和服务端两部分，客户部分主要负责把应用程序的调用信息按一定的格式传送到服务器，并把服务器发回的响应按调用方式返回给应用程序，服务器部分主要接收客户发来的调用请求并完成相应调用，将调用结果返回给客户。

##### 3.1.1 服务器程序

`HelloProc()` 函数由客户端程序中 `doRpcCall()` 函数调用，该函数作用是在服务器端打印出人员的信息，并将打印完成的人员信息的人员编号返回给客户端。

##### 3.1.2 客户端程序

通过 `main()` 函数调用函数 `doRpcCall()`，该函数用于输入人员信息（人员编号、人员姓名、人员备注信息），然后调用服务器程序对数据进行处理，服务器处理完成之后返回处理完成信息。

### 3.2 传输数据结构

本设计传输的数据结构为结构体，如图 3.2:

```
typedef struct PERSON
{
    int code;
    [string] char name[100];
    [string] char note[1024];
}PERSON;
```

图 3.2 自定义的传输数据结构体

定义名为 PERSON 的结构体存储一个人员的信息，其中 code 表示人员的编号，name 表示人员的姓名，note 表示人员的备注信息。



## 4 运行结果

### 4.1 传输成功运行结果

本设计采用 TCP/IP 的网络传输协议，测试时采用两台机器，一台作为服务器，另一台作为客户端，如图 4.1：

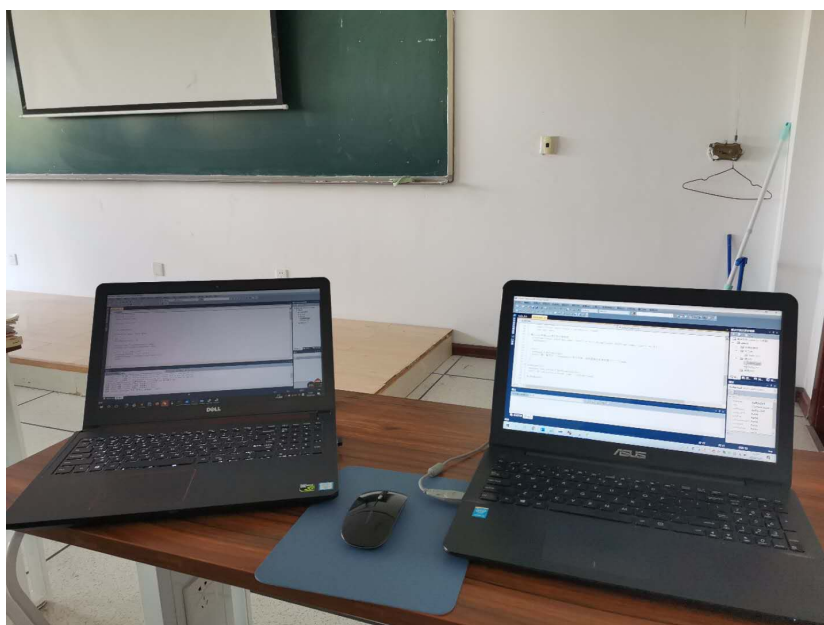


图 4.1 程序运行机器（客户端（左），服务器（右））

测试结果如图 4.2 、4.3：

```
pszStringBinding = neacn_ip_tcp:192.168.1.115[8080]
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')
小明
Please input People's id:
123
Please input People's information:
小明最近表现非常好，非常好！！！！！！！！值得表扬
嘿！编号为:123的小可爱，你的信息已经被处理了！！
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')
小红
Please input People's id:
456
Please input People's information:
小红最近表现非常差，及时督促，及时反思，及时改正
嘿！编号为:456的小可爱，你的信息已经被处理了！！
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')
小王
Please input People's id:
789
Please input People's information:
缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息
缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息
缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息缺少信息
缺少信息缺少信息缺少信息
嘿！编号为:789的小可爱，你的信息已经被处理了！！
```

图 4.2 客户端运行界面



图 4.3 服务器端运行界面

## 4.2 客户端未找到服务器运行

如图 4.4 4.5:

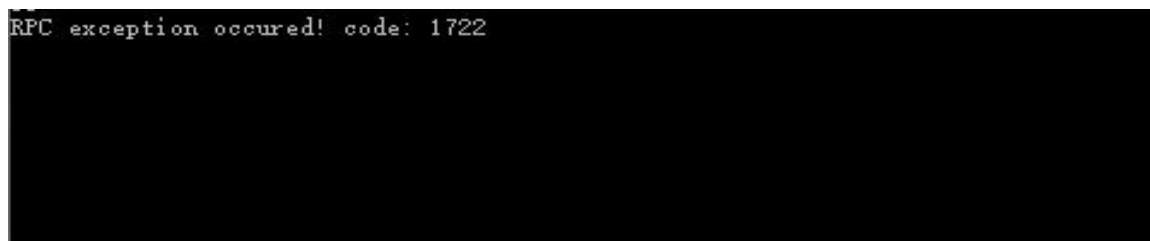


图 4.4 客户端未与服务器连接成功抛出异常

```

“client.exe”: 已加载“C:\Users\ASUS\Desktop\RPC_Advanced Operating System\client\Debug\client.exe”, 已加载符号。
“client.exe”: 已加载“C:\Windows\SysWOW64\ntdll.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\kernel32.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\KernelBase.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\rpcrt4.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\sspicli.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\cryptbase.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\bcryptprimitives.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\sechost.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\ws2_32.dll”, Cannot find or open the PDB file
“client.exe”: 已加载“C:\Windows\SysWOW64\mswsock.dll”, Cannot find or open the PDB file
client.exe 中的 0x74f044c2 处最可能的异常: 0x000006BA: RPC 服务器不可用。
线程“Win32 线程”(0x1fd0)已退出, 返回值为 0 (0x0)。
线程“Win32 线程”(0xa04)已退出, 返回值为 0 (0x0)。
程序“[348] client.exe: 本机”已退出, 返回值为 0 (0x0)。

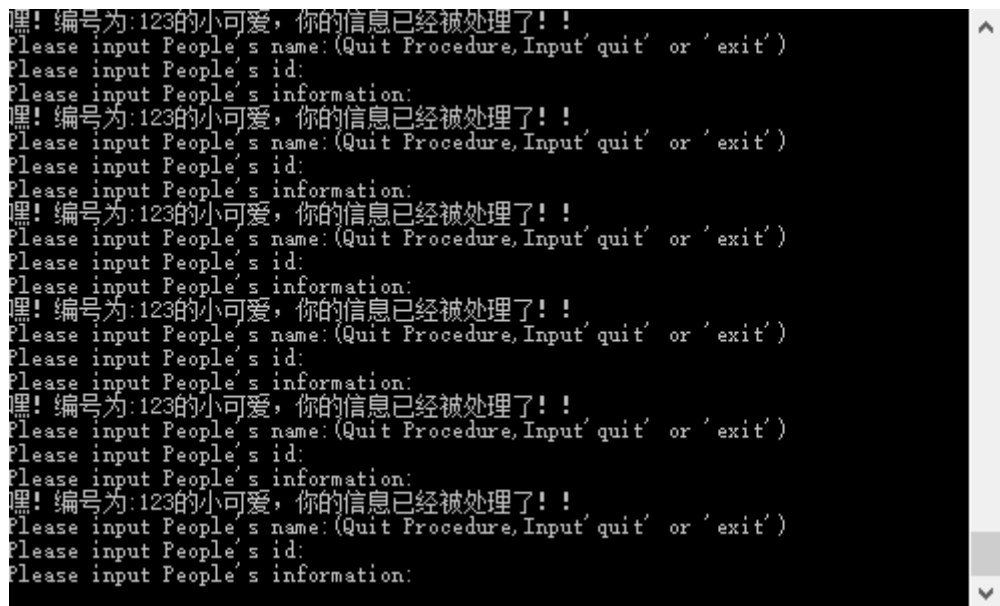
```

图 4.5 客户端未与服务器连接成功 Microsoft Visual Studio 2010 中的调试结果

## 4.3 测试字符串长度超出 1024

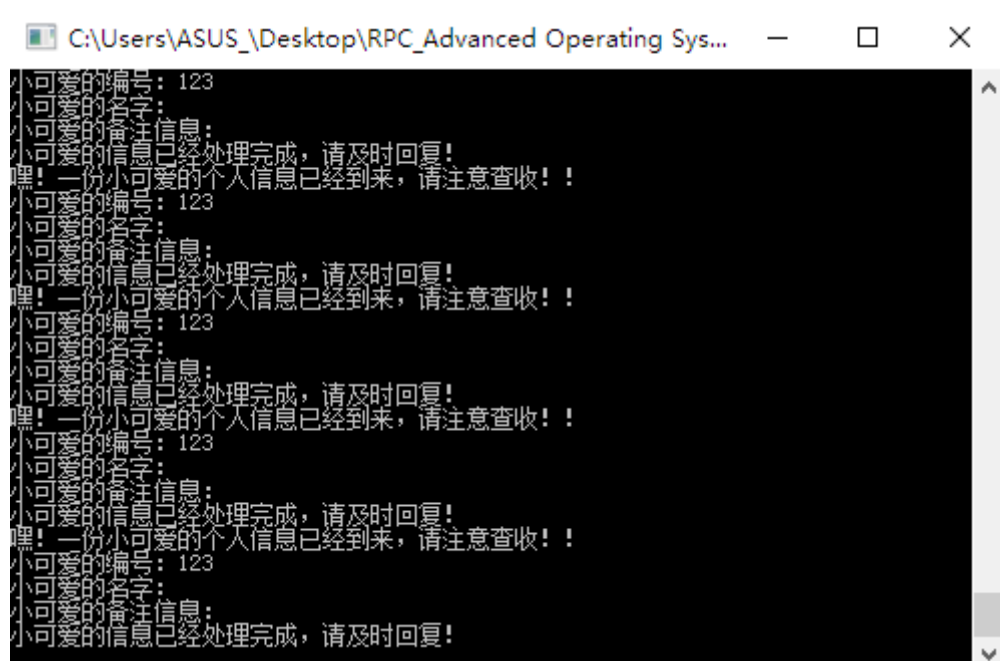
本次设计未对输入字符串长度进行程序处理限定, 仅仅考虑前期设计结构体时,

将其内部的数组变量空间设定的足够大。如果输入字符串长度超出 1024，程序会出错，一直进行循环处理错误情况如图 4.6 4.7:



```
嘿! 编号为:123的小可爱, 你的信息已经被处理了!!  
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')  
Please input People's id:  
Please input People's information:  
嘿! 编号为:123的小可爱, 你的信息已经被处理了!!  
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')  
Please input People's id:  
Please input People's information:  
嘿! 编号为:123的小可爱, 你的信息已经被处理了!!  
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')  
Please input People's id:  
Please input People's information:  
嘿! 编号为:123的小可爱, 你的信息已经被处理了!!  
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')  
Please input People's id:  
Please input People's information:  
嘿! 编号为:123的小可爱, 你的信息已经被处理了!!  
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')  
Please input People's id:  
Please input People's information:  
嘿! 编号为:123的小可爱, 你的信息已经被处理了!!  
Please input People's name: (Quit Procedure, Input 'quit' or 'exit')  
Please input People's id:  
Please input People's information:
```

图 4.6 字符串长度超出 1024 服务器端错误



```
C:\Users\ASUS\Desktop\RPC_Advanced Operating Sys...  
可爱的编号: 123  
可爱的名字:  
可爱的备注信息:  
可爱的信息已经处理完成, 请及时回复!  
嘿! 一份小可爱的个人信息已经到来, 请注意查收!!  
可爱的编号: 123  
可爱的名字:  
可爱的备注信息:  
可爱的信息已经处理完成, 请及时回复!  
嘿! 一份小可爱的个人信息已经到来, 请注意查收!!  
可爱的编号: 123  
可爱的名字:  
可爱的备注信息:  
可爱的信息已经处理完成, 请及时回复!  
嘿! 一份小可爱的个人信息已经到来, 请注意查收!!  
可爱的编号: 123  
可爱的名字:  
可爱的备注信息:  
可爱的信息已经处理完成, 请及时回复!  
嘿! 一份小可爱的个人信息已经到来, 请注意查收!!  
可爱的编号: 123  
可爱的名字:  
可爱的备注信息:  
可爱的信息已经处理完成, 请及时回复!
```

图 4.7 字符串长度超出 1024 服务器端错误

## 5 总结与反思

此次作业历时 4 天，从最开始对 RPC 的知识一无所知，弄懂其原理和实现过程，再到书写出第一个原始程序，对程序的进一步改进，最终实现要求的相关功能，这期间感谢小组内每一位同学的合作与付出。虽然最终实现出的程序功能十分简单，但在这个过程中大家都学习了 RPC 的相关知识，组内每一个成员都积极配合，整个过程大家都发挥了自己的优势，仔细认真完成自己的任务。最后感谢小组内每一位同学的付出。

对于该程序，仍存在以下不足之处：

1. 程序中出现异常，仅仅抛出异常代码，为实现异常的说明工作
2. 客户端输入字符串长度超出 1024，未对其进行程序处理，导致程序出错，后续工作可进行进一步完善
3. 服务器端程序仅仅完成简单的打印和回复人员编号的功能，后续工作可以进行进一步改进
4. 客户端与服务器仅仅实现了一对一的会话操作，为实现服务器端的多线程并发处理功能

## 附录：

### 1.hello.idl 文件

//RPC的接口标准使用了IDL（Interface Description Language接口描述语言）语言标准描述  
//相应微软的编译器是MIDL，通过IDL文件来定义RPC客户端与服务端之间的通信接口，只有通过这些接口客户端才能访问服务器。

```
import "oaidl.idl";  
import "ocidl.idl";
```

//关于uuid:

//UUID是通用唯一识别码（Universally Unique Identifier）的缩写, 是一种软件建构的标准,  
//UUID 的目的是让分布式系统中的所有元素，都能有唯一的辨识资讯，而不需要透过中央控制端来做辨识资讯的指定。

//如此一来，每个人都可以建立不与其它人冲突的 UUID。在这样的情况下，就不需考虑数据库建立时的名称重复问题。

//1. UUID在线生成网站：<https://1024tools.com/uuid>

//2. 或者通过代码自己生成

```
[  
    uuid(296bd6eb-2ca9-4321-875f-5f4cfc10ecbe),  
    version(1.0)  
]
```

//注意： 1)hello.idl文件与hello.acf文件中的接口名称（INTERFACENAME）应一致，否则接下来编译的时候会报错。

//2)hello.idl文件与hello.acf文件应放在同一目录下。

```
interface INTERFACENAME  
{  
    typedef struct PERSON  
    {  
        int code;  
        [string] char name[100];  
        [string] char note[1024];  
    }PERSON;  
    //typedef [ptr] PERSON* PEOPLE;  
    //typedef [ptr] PERSON** PPEOPLE;  
    int HelloProc(PERSON p);  
    void ShutDown(void);  
}
```

```
}
```

//ACF文件可以使用户定义自己的客户端或服务端的RPC接口。  
//例如，如果你的客户端程序包含了一个复杂的数据结构，此数据结构只在本地机上有意义，  
//那么你就可以在ACF文件中指定如何描述独立于机器的数据结构，使用数据结构用于远程过程调用。  
//在ACF文件中定义一个handle类型，用来代表客户端与服务端的连接。[implicit\_handle]属性允许客户端程序为它的远程过程调用选择一个服务端。  
//ACF定义了此句柄为handle\_t类型(MIDL基本数据类型)。MIDL编译器将绑定ACF文件指定的句柄名字hello\_IfHandle，放在生成的头文件hello\_h.h中。

//项目的生成目录下有了hello\_h.h, hello\_c.c, hello\_s.c三个文件。  
//其中，hello\_h.h文件是客户端和服务端程序共同要用到的，  
//hello\_c.c是客户端程序需要的，  
//hello\_s.c是服务器程序所需要的。

## 2.hello.acf 文件

```
[  
implicit_handle (handle_t hello_IfHandle)  
]  
interface INTERFACENAME  
{  
  
}  
}
```

## 3.服务器端 (Server)

//引入rpcrt4.lib依赖包，操作过程为：  
//右键工程属性，在Linker->Input->Additional Dependencies中添加rpcrt4.lib  
//否则无法编译成功

```
#include <iostream>  
using namespace std;  
  
#include "hello_h.h"  
  
int main(void)  
{  
    RPC_STATUS status = 0;
```

```

//RpcServerUseProtseqEp() 详解:
//The RpcServerUseProtseqEp function tells the RPC run-time library to use the specified
protocol sequence combined with the specified endpoint for receiving remote procedure calls.
//RPC_STATUS RpcServerUseProtseqEp(
//RPC_CSTR Protseq, //Pointer to a string identifier of the protocol sequence to
register with the RPC run-time library.
// unsigned int MaxCalls, //Backlog queue length for the ncacn_ip_tcp protocol sequence. All
other protocol sequences ignore this parameter. Use RPC_C_PROTSEQ_MAX_REQS_DEFAULT to
specify the default value.
// RPC_CSTR Endpoint, //Pointer to the endpoint-address information to use in creating
a binding for the protocol sequence specified in the Protseq parameter.
// void *SecurityDescriptor //Pointer to an optional parameter provided for the
security subsystem. Used only for ncacn_np and ncalrpc protocol sequences. All other protocol
sequences ignore this parameter. Using a security descriptor on the endpoint in order to
make a server secure is not recommended. This parameter does not appear in the DCE
specification for this API.
//);

//使用TCP方式作为RPC的通道, 绑定端口号13521
status =
RpcServerUseProtseqEp((RPC_CSTR) "ncacn_ip_tcp", RPC_C_PROTSEQ_MAX_REQS_DEFAULT, (RPC_CSTR)
"8080", NULL);
if(status != 0) {
    cout<<"RpcServerUseProtseqEp returns: "<<status<<endl;
    return -1;
}

// RpcServerRegisterIfEx() 详解:
//The RpcServerRegisterIfEx function registers an interface with the RPC run-time library.
//RPC_STATUS RpcServerRegisterIfEx(
// RPC_IF_HANDLE IfSpec, //MIDL-generated structure indicating the interface to
register. //使用内容在自动生成文件hello_h.h中
// UUID *MgrTypeUuid, //Pointer to a type UUID to associate with the MgrEvp
parameter. Specifying a null parameter value (or a nil UUID) registers IfSpec with a nil-type
UUID.
// RPC_MGR_EPV *MgrEvp, //Manager routines' entry-point vector (EPV). To use the
MIDL-generated default EPV, specify a null value. For more information, please see
RPC_MGR_EPV.
// unsigned int Flags, //Flags. For a list of flag values, see Interface Registration
Flags.
// unsigned int MaxCalls, //If the number of concurrent calls is not a concern, you
can achieve slightly better server-side performance by specifying the default value using
RPC_C_LISTEN_MAX_CALLS_DEFAULT. Doing so relieves the RPC run-time environment from

```

```
enforcing an unnecessary restriction.
// RPC_IF_CALLBACK_FN *IfCallback//Security-callback function, or NULL for no callback.
Each registered interface can have a different callback function. See Remarks for more
details.
//);

//在hello_h.h文件中可以看到hello.idl中所定义的接口实体，一个全局句柄变量（handle_t）
//以及客户端与服务端的接口句柄名INTERFACENAME_v1_0_c_ifspec（客户端使用）和
INTERFACENAME_v1_0_s_ifspec（服务器端使用）。

//从windowsXP SP2开始，增强安全性的要求，如果用的RpcServerRegisterIf()注册接口的话
//客户端调用会出现RpcExceptionCode()==5,即AccessDenied的错误，因此，必须用
//RpcServerRegisterIfEx()带RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH,标志允许客户端直接调用
//status = RpcServerRegisterIf(INTERFACENAME_v1_0_s_ifspec, NULL, NULL);
status =
RpcServerRegisterIfEx(INTERFACENAME_v1_0_s_ifspec, NULL, NULL, RPC_IF_ALLOW_CALLBACKS_WITH
_NO_AUTH, 0, NULL);
if(status != 0){
    cout<<"RpcServerRegisterIf returns: "<<status<<endl;
    return -1;
}

cout<<"Rpc Server Begin Listening..."<<endl;

//RpcServerListen() 详解:
//The RpcServerListen function signals the RPC run-time library to listen for remote
procedure calls. This function will not affect auto-listen interfaces; use
RpcServerRegisterIfEx if you need that functionality.
//RPC_STATUS RpcServerListen(
// unsigned int MinimumCallThreads,//Hint to the RPC run time that specifies the minimum
number of call threads that should be created and maintained in the given server. This value
is only a hint and is interpreted differently in different versions of Windows.
// unsigned int MaxCalls,//Recommended maximum number of concurrent remote procedure calls
the server can execute. To allow efficient performance, the RPC run-time libraries interpret
the MaxCalls parameter as a suggested limit rather than as an absolute upper bound. Use
RPC_C_LISTEN_MAX_CALLS_DEFAULT to specify the default value.
// unsigned int DontWait//Flag controlling the return from RpcServerListen. A value of
nonzero indicates that RpcServerListen should return immediately after completing function
processing. A value of zero indicates that RpcServerListen should not return until the
RpcMgmtStopServerListening function has been called and all remote calls have completed.
//);
status = RpcServerListen(1, 20, FALSE);
if(status != 0){
    cout<<"RpcServerListen returns: "<<status<<endl;
```



```
    return -1;
}

cin.get();
return 0;
}

/*****
//  MIDL malloc & free
//为满足链接需求编写，没有的话会出现连接错误
void * __RPC_USER MIDL_user_allocate(size_t len)
{
    return (malloc(len));
}

void __RPC_USER MIDL_user_free(void*ptr)
{
    free(ptr);
}

/*****
//Interfaces
//为来自hello.idl中的函数

int HelloProc (PERSON p)
{
    //int response;
    cout<<"嘿！一份小可爱的个人信息已经到来，请注意查收！！"<<endl;
    cout<<"小可爱的编号："<<(p).code<<endl;
    cout<<"小可爱的名字："<<(p).name<<endl;
    cout<<"小可爱的备注信息："<<(p).note<<endl;
    cout<<"小可爱的信息已经处理完成，请及时回复！"<<endl;
    return (int)((p).code);
}

void ShutDown(void)
{
    RPC_STATUS status = 0;

    status = RpcMgmtStopServerListening(NULL);
    if(status != 0){
        cout<<"RpcMgmtStopServerListening returns: "<<status<<"!"<<endl;
    }
}
```

```
status = RpcServerUnregisterIf(NULL, NULL, FALSE);
if(status != 0){
    cout<<"RpcServerUnregisterIf returns: "<<status<<"!"<<endl;
}
}
```

## 4.客户端 (Client)

```
#include <iostream>
#include <string>
using namespace std;
```

```
#include "hello_h.h"
```

```
void doRpcCall();
void dealWith();
```

```
int main(int argc, char** argv)
{
    int i = 0;
    RPC_STATUS status = 0;
```

```
    unsigned char * pszNetworkAddr = NULL;
    unsigned char * pszStringBinding = NULL;
```

```
    for(i = 1; i < argc; i++){
        if(strcmp(argv[i], "-ip") == 0){
            pszNetworkAddr = (unsigned char*)argv[++i];
            break;
        }
    }
}
```

//使用TCP方式作为RPC的通道，服务器端口13521（与服务器端一致），第三个参数取NULL则为连接本机服务

//也可以取IP, 域名, severname等

```
status = RpcStringBindingCompose(NULL, (RPC_CSTR)
"ncacn_ip_tcp", (RPC_CSTR)"192.168.1.115", (RPC_CSTR)"8080", NULL, (RPC_CSTR*)&pszStringBin
ding);
if(status != 0){
    cout<<"RpcStringBindingCompose returns: "<<status<<"!"<<endl;
    return -1;
}
```

```

cout<<"pszStringBinding = "<<pszStringBinding<<endl;
status = RpcBindingFromStringBinding((RPC_CSTR)pszStringBinding, &hello_IfHandle);
if(status != 0) {
    cout<<"RpcBindingFromStringBinding returns: "<<status<<"!"<<endl;
    return -1;
}

doRpcCall();

status = RpcStringFree((RPC_CSTR*)&pszStringBinding);
if(status != 0)
    cout<<"RpcStringFree returns: "<<status<<"!"<<endl;

status = RpcBindingFree(&hello_IfHandle);
if(status != 0)
    cout<<"RpcBindingFree returns: "<<status<<"!"<<endl;

cin.get();
return 0;
}

void dealWith(void) {
    PERSON pp;
    int response ;
    while(true) {
        cout<<"Please input People's name:(Quit Procedure, Input 'quit' or 'exit')"<<endl;
        cin>>(pp).name;
        //输入exit或者quit退出客户端程序
        if(strcmp((const char*)(pp).name, "exit") == 0 || strcmp((const char*)(pp).name,
"quit") == 0) {
            ShutDown();
        }
        else{
            cout<<"Please input People's id:"<<endl;
            cin>>(pp).code;
            cout<<"Please input People's information:"<<endl;
            cin>>(pp).note;
            response = HelloProc(pp);
            cout<<"嘿! 编号为:"<<response<<"的小可爱, 你的信息已经被处理了!! "<<endl;
        }
    }
}

```

//客户端输入字符程序

```
void doRpcCall(void)
{
    RpcTryExcept{
        dealWith();
    }
    RpcExcept(1){
        unsigned long ulCode = RpcExceptionCode();
        cout<<"RPC exception occurred! code: "<<ulCode<<endl;
    }
    RpcEndExcept
}
```

//为满足链接需求编写，没有的话会出现连接错误

```
void * __RPC_USER MIDL_user_allocate(size_t len)
{
    return (malloc(len));
}
```

```
void __RPC_USER MIDL_user_free(void* ptr)
{
    free(ptr);
}
```