一种 Windows 平台下本地进程间过程调用的实现方法及其应用①

肖利平 (91655 部队 北京 100036)

摘　要： 在实际的应用软件开发工作中可能需要为下面的情形提供一种解决方案：有一簇结构非常复杂的功能函数集，有若干个不同的应用都需要利用这一簇功能集来实现相应的功能。如何完成上述情形的开发工作，有很多可选方案。提出了一种进程间过程调用的实现方法，并就如何把此方法应用于上述问题的解决以及在实际开发中应该注意的问题进行了阐述。

关键词： Windows 平台；内存映射文件；进程间过程调用

Realization and Application of the Local Process Internal Procedure Call to Windows Platform

XIAO Li-Ping

(Unit 91655 of People's Liberation Army of China, Beijing 100036, China)

Abstract: It often happens in software development that several different applications utilize the same set of sophisticated functions to achieve corresponding roles. There are many options to accomplish the task aforementioned. This paper proposes herein an approach to realize the Local Process Internal Procedure Call. It further elucidates its application in the above situation as well as some tricks in practical development.

Keywords: Windows platform; memory mapping file; process internal procedure call

1　引言

在实际的应用软件开发工作中我们可能需要为下面的情形提供一种解决方案：有一簇结构非常复杂的功能函数集，有若干个不同的应用都需要利用这一簇功能集的一部分来实现相应的功能。通常，我们可以采用以下三种不同形式的解决方案：

1) 所有的应用分别进行开发，互不干扰也没有共享；

2) 把这一簇功能集以控件的形式进行封装，再由其他应用来使用；

3) 把各个应用需要使用的这一簇功能集的功能集中起来，单独进行开发，最终形成一个功能完备的应用程序，由这个应用程序为其他的应用提供接口和服务，其他的应用通过这个应用程序提供的接口来使用其所提供的服务。

以上三种方案各有利弊，具体采用那种方案要视实际情况而定。以上三种方案可以用图 1 来说明。

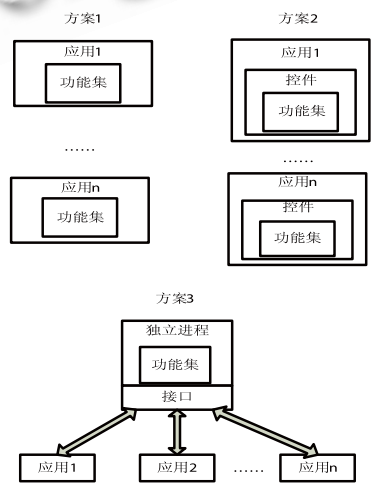

图 1　3 种不同的解决方案

① 收稿时间:2009-07-20;收到修改稿时间:2009-09-08

2 进程间过程调用

所谓进程间过程调用，是指下面的情形：有两个进程 A 和 B，其中进程 A 以函数集的形式实现了一组功能，进程 B 以过程调用的形式来请求进程 A 执行某项功能，当进程 A 接到请求后，利用进程 B 传递过来的参数，然后调用函数集中的函数来实际的操作，并把相关返回值(如果有)返回给进程 B，这样就完成了一次进程 B 到进程 A 的过程调用，我们把进程 A 称为服务提供进程，进程 B 称为客户进程。

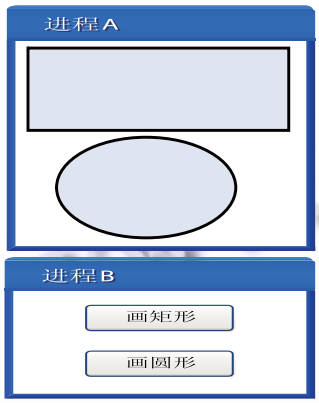

图 2 进程间过程调用示例

例如，图 2 显示了两个进程 A 和 B 的 GUI，进程 A 实现了一组图形绘制功能，当点击进程 B 的“画矩形”按钮，进程 B 会调用函数 DrawRectangle(10，100， 210， 150)，但是进程 B 中的这个函数并没有实现具体的矩形绘制功能，而只是把有关参数传递给进程 A，然后由进程 A 调用相关的函数来实现矩形的绘制，并把有关返回值返回给进程 B。从表现形式上看，进程 A 的窗口就像是进程 B 的一个视图窗口，从实现上看，对于进程 B，与同一个进程内的普通函数调用几乎没有什么差别。以上用图形绘制函数的调用为例说明了进程间过程调用的过程和表现形式，其它功能函数的调用具有类似的过程和表现形式。

3 实现方法

为了在 Windows 平台上实现如上所述的进程间过程调用，需要解决两个问题：1)参数如何传递；2)如何定位要调用的函数，也即服务提供进程如何确定要调用的函数是那一个函数。

3.1 参数传递问题

在 Windows 平台上，进程间可以快速和方便的共享数据和信息。Win32 和 16 位 Windows 都能用若干方法来完成这些任务[1]。例如，在 16 位 Windows 里，可以使用 GMEM_SHARE 标志来分配一块全局内存，再调用 SendMessage 或 PostMessage 来传送句柄(作为参数 wParam 或 lParam)[1]。消息的接收者再调用 GlobalLock 来得到内存块的地址，就可以读写数据了[1]。不过这种方法在 Win32 中是行不通的，因为每个进程有自己的地址空间，一个进程不能轻易地访问另一个进程地址空间中的数据[1]。那么在 Win32 如何实现多个进程间的数据共享呢？答案就是通过使用内存映射文件。实际上，在 Win32 中，内存映射文件是多个进程共享数据的唯一机制[1]。 因此对于参数传递问题我们可以通过使用内存映射文件来解决。关于内存映射文件，有必要对 CreateFileMapping 和 MapViewOfFile 两个 Windows API 函数进行简单的说明。

CreateFileMapping 的原型为：

```
HANDLE CreateFileMapping(
  HANDLE hFile,
  LPSECURITY_ATTRIBUTES lpAttributes,
DWORD flProtect,
  DWORD dwMaximumSizeHigh,
  DWORD dwMaximumSizeLow,
  LPCTSTR lpName );
```

MapViewOfFile 的原型为：

```
LPVOID MapViewOfFile(
  HANDLE hFileMappingObject,
  DWORD dwDesiredAccess,
  DWORD dwFileOffsetHigh,
  DWORD dwFileOffsetLow,
  SIZE_T dwNumberOfBytesToMap );
```

在这里需要指出的是，当参数 hFile 的值为 INVALID_HANDLE_VALUE(0xFFFFFFFF) 时， CreateFileMapping 函数将创建由系统的页面文件支持的内存映射文件，而不是物理存储位于磁盘上的文件的文件映射对象[1]。分配的物理存储的数量是由参数 dwMaximumSizeHigh 和 dwMaximumSizeLow 决定的。

在创建了该文件映射对象，并将它的一个视图映

射到进程地址空间中(通过调用 MapViewOfFile 函数)之后，就能像使用任意内存区域一样使用它了[1]。如果要同其他进程共享此数据，可以在调用 CreateFileMapping 时传递一个字符串作为 lpName 参数。其他想要访问该存储的进程可以传递同一名字来调用 CreateFileMapping[1]。

通过内存映射文件解决了数据共享问题，对于参数传递，还需要解决参数的识别问题，也即参数的存储格式设计问题。在设计参数的存储格式时，需要考虑以下几个方面的问题：

参数是否定长(也即参数占用的字节数是否固定)，对于定长参数(例如 int，char 等等)比较好办，对于不定长参数(例如字符串)还必须添加参数长度描述信息；

参数为指针类型，这时必须分清楚参数是输入参数还是输出参数，必须区别对待，同时还要注意指针所指内容是否定长；

参数为结构类型，这时需要弄清楚结构体中是否包含不定长成员，是否包含指针成员等等。

以上列举的只是笔者在实际工作中，关于参数存储格式设计时所碰到的一些问题，具体的设计方法，由于篇幅问题，这里就不再赘述。

3.2 关于调用函数定位问题

对于服务提供进程来说，可以用如下的方式来实现，每一个函数对应一个序号，客户进程通过传递一个序号来确定要调用的函数。服务提供进程的实现形式如下所示：

```
switch (iIndex){
case 1:
获取参数；
调用函数 1；
case 2:
获取参数；
调用函数 2；
…
}
```

如果只有三五个函数采用这种方式还无可非议，但如果有几十上百个函数，还用这种方式来实现的话，只能用“丑陋”来形容了，基本没有可维护性可言，为此笔者提出了如下解决方案。

在本解决方案中，实现了三个动态链接库：DllA.dll、DllB.dll 和 DllC.dll。其结构如图 3 所示。

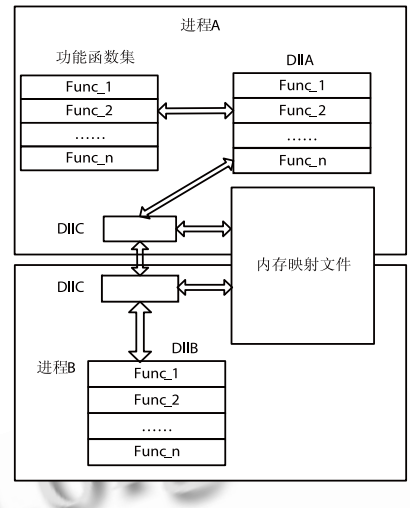

图 3　结构图

其中，DllA.dll 对服务提供进程 A 所提供的功能函数集进行包装，其主要功能是对传递过来的参数进行解析，然后调用功能函数集中对应的函数。在 DllA.dll 中所有函数都采用如下的形式。

```
void  Func(void){
    解析参数；
    调用功能函数集中相应的函数；
    把返回值写入内存映射文件；
}
```

DllC.dll 实现进程间的通信，负责内存映射文件的管理。当进程 A 调用 DllC.dll 中的初始化函数时，DllC.dll 会调用 g_hModule = LoadLibrary(“DllA.dll”)来载入动态链接库 DllA.dll，调用 CreateFileMapping 函数创建内存映射文件，并在进程 A 中创建一个隐藏的对话框，进程 A 和进程 B 共享该对话框的窗口的句柄。该对话框的窗口过程具有如下的实现形式：

```
BOOL WINAPI DlgWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
        ……
if(uMsg == MSG_CALLFUNC) {
    //从内存映射文件中读取被调用函数的函数 lpszFuncName
        CallFunc(lpszFuncName);
        }
        ……
        return FALSE;
```

```
}
DWORD    CallFunc(LPCTSTR lpzFuncName) {
    //获取 DllA.dll 中被调函数的地址
    FARPROC fp = GetProcAddress(g_ hModule, lpzFuncName);
    //调用函数
    if(0 != fp)  {
        fp();
    }
    return 0;
}
```

DllB.dll 用来包装提供给客户进程的接口，负责把调用的函数名以及调用参数写入内存映射文件，并通知进程 A，然后从内存映射文件中获取返回值，并返回给调用者。DllB.dll 中的函数与功能函数集中的函数一一对应，例如功能函数集中有函数 int DoSomthing(int I, int j)，那么在在 DllB.dll 中相应的有函数 int DllB_DoSomthing(int I, int j)，只不过函数 int DllB_ DoSomthing(int I, int j)并不实现具体的功能，而只是把相关的调用数据(调用函数名和调用参数)写入内存映射文件，并通知进程 A，然后进程 A 再调用函数然 int DoSomthing(int I, int j)来实现实际的功能，并把返回值写入内存映射文件，最后进程 B 从内存映射文件中读取返回值并返回给调用者。DllB.dll 中函数具有如下的形式：

返回值 DllB_Func(参数列表) {

把被调用的 DllA.dll 中函数的函数名写入内存映射文件；

把参数写入内存映射文件；

//通知进程 A(通过调用 DllC.dll 中的 SendMsg()函数)

SendMsg(MSG_CALLFUNC)

从内存映射文件中读取返回值；

返回；

}

进程 B 调用进程 A 中的函数的具体过程为：

int DoSomeThing(int i, int j) {

① 把要调用的 DllA 中的函数的名称写入内存映射文件；

② 把调用参数(如果有)写入内存映射文件；

③ 调用 DllC(在进程 B 中)中的 SendMsg 函数通知进程 A；

以下处理在进程 A 中进行

④ 进程 A 收到通知(DllC 中的 DlgWndProc)；

从共享内存中获取要调用的 DllA 中的函数的名称 lpzFuncName；

获取 DllA.dll 中被调函数的地址

FARPROC fp = GetProcAddress(g_hModule, lpzFuncName);

调用函数：

fp(){

解析参数(int i, int j)；

调用功能函数集中的函数：

int iRet = DoSomeTing(i, j);

返回值(iRet)写入内存映射文件；

返回；

}

进程 A 处理结束；

⑤ 从内存映射文件中读取返回值；

返回；

}

4 结语

从笔者的实际工作经验来看，利用进程间过程调用方法来解决本文开头所提出的问题，是一种非常经济的方案，虽然在实际开发上存在一定的难度，但是这种方式优越性还是很显著的。首先，利用这种方式可以减少很多不必要的重复劳动；其次，采用这种方式使系统具有很好的可扩展性；第三，采用这种方式可以有效地隔离功能函数集的修改，使得功能函数集的修改不会波及到其它的应用。

当然，在实际工作中，除了要解决本文提出的一些问题之外，还有很多其它问题需要解决，例如如果在同一台机器上必须运行多个应用，那么还必须解决不同应用的标识以及相关的管理问题；此外，还有线程同步问题等等。

对于本文存在的不足之处，还望各位读者朋友不吝赐教。

参考文献

1 Richter J.王书洪,刘光明 译.Windows 高级编程指南. 北京:清华大学出版社, 1999.