**摘 要:** 随着人们对应用程序的要求越来越高，单进程应用在许多场合已不能满足人们的要求。编写多进程/多线程程序成为现代程序设计的一个重要特点，在多进程程序设计中，进程间的通信是不可避免的。Microsoft Win32 API提供了多种进程间通信的方法，全面地阐述了这些方法的特点，并加以比较和分析，希望能给读者选择通信方法提供参考。  
**关键词** 进程 进程通信 IPC Win32 API

**1 进程与进程通信**

　　进程是装入内存并准备执行的程序，每个进程都有私有的虚拟地址空间，由代码、数据以及它可利用的系统资源(如文件、管道等)组成。多进程/多线程是Windows操作系统的一个基本特征。Microsoft Win32应用编程接口(Application Programming Interface, API)提供了大量支持应用程序间数据共享和交换的机制，这些机制行使的活动称为进程间通信(InterProcess Communication, IPC)，进程通信就是指不同进程间进行数据共享和数据交换。  
　　正因为使用Win32 API进行进程通信方式有多种，如何选择恰当的通信方式就成为应用开发中的一个重要问题，下面本文将对Win32中进程通信的几种方法加以分析和比较。

**2 进程通信方法**

**2.1 文件映射**  
　　文件映射(Memory-Mapped Files)能使进程把文件内容当作进程地址区间一块内存那样来对待。因此，进程不必使用文件I/O操作，只需简单的指针操作就可读取和修改文件的内容。  
　　Win32 API允许多个进程访问同一文件映射对象，各个进程在它自己的地址空间里接收内存的指针。通过使用这些指针，不同进程就可以读或修改文件的内容，实现了对文件中数据的共享。  
　　应用程序有三种方法来使多个进程共享一个文件映射对象。  
　　(1)继承：第一个进程建立文件映射对象，它的子进程继承该对象的句柄。  
　　(2)命名文件映射：第一个进程在建立文件映射对象时可以给该对象指定一个名字(可与文件名不同)。第二个进程可通过这个名字打开此文件映射对象。另外，第一个进程也可以通过一些其它IPC机制(有名管道、邮件槽等)把名字传给第二个进程。  
　　(3)句柄复制：第一个进程建立文件映射对象，然后通过其它IPC机制(有名管道、邮件槽等)把对象句柄传递给第二个进程。第二个进程复制该句柄就取得对该文件映射对象的访问权限。  
　　文件映射是在多个进程间共享数据的非常有效方法，有较好的安全性。但文件映射只能用于本地机器的进程之间，不能用于网络中，而开发者还必须控制进程间的同步。

**2.2 共享内存**  
　　Win32 API中共享内存(Shared Memory)实际就是文件映射的一种特殊情况。进程在创建文件映射对象时用0xFFFFFFFF来代替文件句柄(HANDLE)，就表示了对应的文件映射对象是从操作系统页面文件访问内存，其它进程打开该文件映射对象就可以访问该内存块。由于共享内存是用文件映射实现的，所以它也有较好的安全性，也只能运行于同一计算机上的进程之间。

**2.3 匿名管道**  
　　管道(Pipe)是一种具有两个端点的通信通道：有一端句柄的进程可以和有另一端句柄的进程通信。管道可以是单向－一端是只读的，另一端点是只写的；也可以是双向的－管道的两端点既可读也可写。  
　　匿名管道(Anonymous Pipe)是 在父进程和子进程之间，或同一父进程的两个子进程之间传输数据的无名字的单向管道。通常由父进程创建管道，然后由要通信的子进程继承通道的读端点句柄或写 端点句柄，然后实现通信。父进程还可以建立两个或更多个继承匿名管道读和写句柄的子进程。这些子进程可以使用管道直接通信，不需要通过父进程。  
　　匿名管道是单机上实现子进程标准I/O重定向的有效方法，它不能在网上使用，也不能用于两个不相关的进程之间。

**2.4 命名管道**  
　　命名管道(Named Pipe)是服务器进程和一个或多个客户进程之间通信的单向或双向管道。不同于匿名管道的是命名管道可以在不相关的进程之间和不同计算机之间使用，服务器建立命名管道时给它指定一个名字，任何进程都可以通过该名字打开管道的另一端，根据给定的权限和服务器进程通信。  
　　命名管道提供了相对简单的编程接口，使通过网络传输数据并不比同一计算机上两进程之间通信更困难，不过如果要同时和多个进程通信它就力不从心了。

**2.5 邮件槽**  
　　邮件槽(Mailslots)提供进程间单向通信能力，任何进程都能建立邮件槽成为邮件槽服务器。其它进程，称为邮件槽客户，可以通过邮件槽的名字给邮件槽服务器进程发送消息。进来的消 息一直放在邮件槽中，直到服务器进程读取它为止。一个进程既可以是邮件槽服务器也可以是邮件槽客户，因此可建立多个邮件槽实现进程间的双向通信。  
　　通过邮件槽可以给本地计算机上的邮件槽、其它计算机上的邮件槽或指定网络区域中所有计算机上有同样名字的邮件槽发送消息。广播通信的消息长度不能超过400字节，非广播消息的长度则受邮件槽服务器指定的最大消息长度的限制。  
　　邮件槽与命名管道相似，不过它传输数据是通过不可靠的数据报(如TCP/IP协议中的UDP包)完成的，一旦网络发生错误则无法保证消息正确地接收，而命名管道传输数据则是建立在可靠连接基础上的。不过邮件槽有简化的编程接口和给指定网络区域内的所有计算机广播消息的能力，所以邮件槽不失为应用程序发送和接收消息的另一种选择。

**2.6 剪贴板**　　剪贴板(Clipped Board)实质是Win32 API中一组用来传输数据的函数和消息，为Windows应用程序之间进行数据共享提供了一个中介，Windows已建立的剪切(复制)－粘贴的机制为不同应用程序之间共享不同格式数据提供了一条捷径。当用户在应用程序中执行剪切或复制操作时，应用程序把选取的数据用一种或多种格式放在剪贴板上。然后任何其它应用程序都可以从剪贴板上拾取数据，从给定格式中选择适合自己的格式。  
　　剪贴板是一个非常松散的交换媒介，可以支持任何数据格式，每一格式由一无符号整数标识，对标准(预定义)剪贴板格式，该值是Win32 API定义的常量；对非标准格式可以使用Register Clipboard Format函数注册为新的剪贴板格式。利用剪贴板进行交换的数据只需在数据格式上一致或都可以转化为某种格式就行。但剪贴板只能在基于Windows的程序中使用，不能在网络上使用。

**2.7 动态数据交换**  
　　动态数据交换(DDE)是使用共享内存在应用程序之间进行数据交换的一种进程间通信形式。应用程序可以使用DDE进行一次性数据传输，也可以当出现新数据时，通过发送更新值在应用程序间动态交换数据。  
　　DDE和剪贴板一样既支持标准数据格式(如文本、位图等)，又可以支持自己定义的数据格式。但它们的数据传输机制却不同，一个明显区别是剪贴板操作几乎总是用作对用户指定操作的一次性应答－如从菜单中选择Paste命令。尽管DDE也可以由用户启动，但它继续发挥作用一般不必用户进一步干预。DDE有三种数据交换方式：  
　　(1) 冷链：数据交换是一次性数据传输，与剪贴板相同。  
　　(2) 温链：当数据交换时服务器通知客户，然后客户必须请求新的数据。  
　　(3) 热链：当数据交换时服务器自动给客户发送数据。  
　　DDE交换可以发生在单机或网络中不同计算机的应用程序之间。开发者还可以定义定制的DDE数据格式进行应用程序之间特别目的IPC，它们有更紧密耦合的通信要求。大多数基于Windows的应用程序都支持DDE。

**2.8 对象连接与嵌入**  
　　应用程序利用对象连接与嵌入(OLE)技术管理复合文档(由多种数据格式组成的文档)，OLE提供使某应用程序更容易调用其它应用程序进行数据编辑的服务。例如，OLE支持的字处理器可以嵌套电子表格，当用户要编辑电子表格时OLE库可自动启动电子表格编辑器。当用户退出电子表格编辑器时，该表格已在原始字处理器文档中得到更新。在这里电子表格编辑器变成了字处理器的扩展，而如果使用DDE，用户要显式地启动电子表格编辑器。  
　　同DDE技术相同，大多数基于Windows的应用程序都支持OLE技术。

**2.9 动态连接库**  
　　Win32动态连接库(DLL)中的全局数据可以被调用DLL的所有进程共享，这就又给进程间通信开辟了一条新的途径，当然访问时要注意同步问题。  
　　虽然可以通过DLL进行进程间数据共享，但从数据安全的角度考虑，我们并不提倡这种方法，使用带有访问权限控制的共享内存的方法更好一些。

**2.10 远程过程调用**  
　　Win32 API提供的远程过程调用(RPC)使应用程序可以使用远程调用函数，这使在网络上用RPC进行进程通信就像函数调用那样简单。RPC既可以在单机不同进程间使用也可以在网络中使用。  
　　由于Win32 API提供的RPC服从OSF-DCE(Open Software Foundation Distributed Computing Environment)标准。所以通过Win32 API编写的RPC应用程序能与其它操作系统上支持DEC的RPC应用程序通信。使用RPC开发者可以建立高性能、紧密耦合的分布式应用程序。

**2.11 NetBios函数**  
　　Win32 API提供NetBios函数用于处理低级网络控制，这主要是为IBM NetBios系统编写与Windows的接口。除非那些有特殊低级网络功能要求的应用程序，其它应用程序最好不要使用NetBios函数来进行进程间通信。

**2.12 Sockets**  
　　Windows Sockets规范是以U.C.Berkeley大学BSD UNIX中流行的Socket接口为范例定义的一套Windows下的网络编程接口。除了Berkeley Socket原有的库函数以外，还扩展了一组针对Windows的函数，使程序员可以充分利用Windows的消息机制进行编程。  
　　现在通过Sockets实现进程通信的网络应用越来越多，这主要的原因是Sockets的跨平台性要比其它IPC机制好得多，另外WinSock 2.0不仅支持TCP/IP协议，而且还支持其它协议(如IPX)。Sockets的唯一缺点是它支持的是底层通信操作，这使得在单机的进程间进行简单数据传递不太方便，这时使用下面将介绍的WM\_COPYDATA消息将更合适些。

**2.13 WM\_COPYDATA消息**  
　　WM\_COPYDATA是一种非常强大却鲜为人知的消息。当一个应用向另一个应用传送数据时，发送方只需使用调用SendMessage函数，参数是目的窗口的句柄、传递数据的起始地址、WM\_COPYDATA消息。接收方只需像处理其它消息那样处理WM\_COPY DATA消息，这样收发双方就实现了数据共享。  
　　WM\_COPYDATA是一种非常简单的方法，它在底层实际上是通过文件映射来实现的。它的缺点是灵活性不高，并且它只能用于Windows平台的单机环境下。

**3 结束语**

　　Win32 API为应用程序实现进程间通信提供了如此多种选择方案，那么开发者如何进行选择呢？通常在决定使用哪种IPC方法之前应考虑以下一些问题：  
　　(1)应用程序是在网络环境下还是在单机环境下工作。

**附:**

在我学windows编程的时候，对进程间如何通信总是感觉很神秘，网络上介绍的方法很多，但是很少有一个系统的介绍，五花八门的说法让人总是一头雾水，在这里，我整理一下各通信方法，梳理了一下这些方法的优缺点，希望能对各位看官起到抛砖引玉的作用。 非标准的进程间通信技术有：Windows消息，内存映射，内存共享等等。

1. Windows消息实现进程间通信。 消息的接受进程和发送进程都要定义相同的消息。但是，如果发送方仅仅是发送消息，那么发送方可以不实现消息映射，不用定义消息响应函数。接受方需要定义映射和相应函数。 自定义消息的实现进程间通信的缺陷是，由于消息的传递参数是个长整形 lparam， 因此，只能传递整形的数据，而不能传递字符串。有个可以的思路是，传递字符串所在的地址，然后另一个程序通过获得发送方的进程句柄，用函数ReadProcessMemory与WriteProcessMemory操作发送方的内存空间来读取内容。

2. 使用MFC定义的WM\_COPYDATA消息 该消息其实与普通的自定义消息通信类似，区别是，传送的是一个指向COPYDATASTRUCT 结构的指针，要传递的字符串就保存在这个结构体里。这个结构的第一个变量 dwData可以设置为0即可。 这个消息与 上面的那个思路的不同是，获得了结构体的指针后，接受进程不需要其他的处理就能获取到指针的数据，就好像在同一个进程里通信一样. 另外注意，该消息的接受有时并不能获得所需要的长度，有可能只接收到了一部分。 因此，该方式只适合于传递是少量的数据。

3.使用进程内存读写函数 基本上与方法一得后面猜想部分相同。关键是，要使用GlobalAlloc()或者VirtualAllocEx来分配内存空间存放数据。把数据写入接收方的进程内存，然后接着就发消息告诉他数据的地址。由于是在发送方申请的内存，那么，最好等待sleep一定时间再VirtualFreeEx申请到得内存。 GlobalAlloc()或者VirtualAllocEx可以实现在另一个进程的内存空间来申请内存，这就为什么上面能进行的原因。也就是说，发送方并不在自己的内存空间申请内存，而是在接收方进程内存空间来申请内存。然后写入输入数据。当然，也可以在发送方的进程空间来申请内存，接收方通过跨进程读写的方式来读。这只是处理方式不同而已。

4.使用内存映射文件的方法 内存映射的好处就是，可以像对待一个文件一样来对待一块内存区。文件时可以被不同的进程来读写的。既然那块内存 区像文件一样，那么这块内存区就可以被不同的进程来读写。

5.使用DLL进行通信 Win32 DLL 只能共享代码不能共享数据，不同的进程载入同一个DLL文件，DLL的代码都只载入了一份到内存，这只载入一份代码仅指同一个DLL文件，如果相同的DLL文件在不同的盘符下，也不是同一个DLL，而是同一DLL多个副本。 Win16 DLL 被载入了系统内存，所以调用它的进程都可以访问到它的全局变量，因此可以很容易的实现进程间通信。但是对于win32 DLL , 操作系统会把该DLL映射到每个调用它的进程的地址空间，DLL成为该进程的一部分。 可以用下面的方法来将DLL的数据区设置为共享区。

#pragma data\_seg("SHARED") // 定义名为SHARED的共享数据段

char m\_strString[256]=TEXT(""); // 共享的数据。特别要注意需要初始化，因为编译器会把未初始化的变量保存在bss数据段。

volatile bool bInCriticalSection=FALSE; // 同步标示

#pragma data\_seg()

 #pragma comment(linker,"/SECTION:SHARED,RWS") // 将要共享的数据段通知编译器。

CCriticalSection cs; // 临界区，控制同步的 上面控制了同步问题。

6.使用操作系统提供的剪贴板实现通信 使用剪贴板是一中开销较小的进程通信机制。剪贴板机制的原理是，剪贴板是系统预留的一块全局内存，用来暂存进程间进行数据交换的数据。 提供数据的进程需要先创建一个全局内存块，然后将要传送的数据移到或复制到该内存块。 接收进程需要获得此内存块的句柄，完成数据读取。

下面的程序标明怎么在剪贴板写数据

CString strData=m\_strClipBoard; // 获得数据.

// 打开系统剪贴板.

if (!OpenClipboard()) return;

// 使用之前，清空系统剪贴板.

EmptyClipboard();

// 分配一内存，大小等于要拷贝的字符串的大小，返回该内存控制句柄.

HGLOBAL hClipboardData;

hClipboardData = GlobalAlloc(GMEM\_DDESHARE, strData.GetLength()+1);

 // 内存控制句柄加锁，返回值为指向那内存控制句柄所在的特定数据格式的指针.

char \* pchData; pchData = (char\*)GlobalLock(hClipboardData);

// 将本地变量的值赋给全局内存.

strcpy(pchData, LPCSTR(strData));

 // 给加锁的全局内存控制句柄解锁.

GlobalUnlock(hClipboardData);

// 通过全局内存句柄将要拷贝的数据放到剪贴板上.

SetClipboardData(CF\_TEXT,hClipboardData);

// 使用完后关闭剪贴板.

CloseClipboard();

从剪贴板读取数据的代码如下：

// 打开系统剪贴板.

if (!OpenClipboard()) return;

// 判断剪贴板上的数据是否是指定的数据格式.

if (IsClipboardFormatAvailable(CF\_TEXT)|| IsClipboardFormatAvailable(CF\_OEMTEXT)) {

// 从剪贴板上获得数据.

HANDLE hClipboardData = GetClipboardData(CF\_TEXT);

// 通过给内存句柄加锁，获得指向指定格式数据的指针.

char \*pchData = (char\*)GlobalLock(hClipboardData);

 // 本地变量获得数据.

m\_strClipBoard = pchData;

// 给内存句柄解锁.

GlobalUnlock(hClipboardData);

}

else

 {

 AfxMessageBox("There is no text (ANSI) data on the Clipboard."); }

// 使用完后关闭剪贴板.

CloseClipboard();

// 更新数据.

 UpdateData(FALSE);

7.DDE （Dynamic Data Exchange）动态数据交换 目前微软已经停止了开发这种技术，仅仅保留了支持。 高级通信技术 前面都是几种基本的进程通信技术，消息管道（Message Pipes）, 邮槽（Mail slots）， 和套接字（Sockets）则是实际比较常见的方法，这几种高级通信除了可以像上面的方法样实现本地系统进程间的通信，也可以用于远程不同系统间的通信。

8. 消息管道 Message Pipes又分为匿名管道(anonymous Pipes),和 命名管道（Named Pipes）, 匿名管道主要用于本地系统上父进程与他启动的子进程间的通信。命名管道高级些，可以再不同的系统上的进程间通信，因为UNIX, LINUX等都支持这项技术，因此，命名管道技术是比较理想的C/S结构通信技术。 命名管道原理是，一个进程把数据放进管道中，另一个知道管道名字的经常来把数据取走。其他不知道管道名字的进程不可能能把数据取走。 因此，管道实际上就是一块进程间的共享内存。 创建管道的进程叫管道服务器，链接管道的进程就是客户机。创建管道的函数是HANDLE CreateNamedPipe(….); 连接管道 CallNamedPipe(); 读入或写入数据后腰关闭它 ConnectNamedPipe(); 服务器准备好一个连接到客户进程的管道，并一直等待知道客户连接上为止。 DisconnectNamedPipe(); 服务器中断与客户的连接 GetNamedPipeHandleState(); 获取一个命名管道的状态信息 GetNamedPipeInfo();获取一个命名管道的信息 PeekNamedPipe(); 从一个管道中复制数据到一个缓冲区 SetNamedPipeHandleState(); 设置一个管道的状态信息以及管道类型 TransactNamedPipe(); 从一个消息管道读消息或写消息 WaitNamedPipe(); 使服务器等待来自客户的实例连接。 管道通信基本流程

（1）连接建立 服务器端通过ConnectNamedPipe函数建立以个命名管道实例，然后通过ConnectNamedPipe()函数来侦听来自客户端得请求。这个函数可以设置等待时间。 客户端只需使用函数WaitNamedPipe()来连接服务器。

（2）通信实现 建立连接后，就可以通过得到管道的文件句柄利用ReadFile()与WriteFile()进行彼此间的通信。

（3）连接终止 客户端调用CloseFile(), 服务器端调用DisconnectNamedPipe()终止连接。且都需要CloseHandle来关闭管道。

9.邮槽通信

10.套接字通信 套接字通信过程可简单的描述为，主要调用5个函数，socket(),bind(), Listen(), connect(), accept(). 服务器端主要调用socket(),bind(), Listen(),accept()。 客户端主要调用socket(),connect()。 双方的数据传送就是通过send() 与recv()完成。 套接字类型主要有5种，SOCK\_STREAM, SOCK\_DGRAM, SOCK\_RAW, SOCK\_SEQPACKET, SOCK\_RDM.

10.1 Winsock 程序设计流程

（1）程序编译环境 有两套函数进行Winsock程序设计，Socket1.1 与Socket2.0. 可以灵活混用。Socket2.0得功能较为强大。 包含其中一个头文件及其对应的库文件即可。

// Socket2.0

#include

#pragma comment (lib, “ws2\_32.lib”);

// Socket1.1

#include

#pragma comment (lib, “wsock32.lib”);

（2）选择机制（异步？非阻塞？） 默认情况下都是创建的阻塞套接字。可以通过select或者WSAAsynSelect()函数将其变为非阻塞的。特别注意，用这个函数改为非阻塞后，不能简单的利用ioctlsocket()将它再变为阻塞模式。也就是说这两个函数改变阻塞模式是由区别的。ioctlsocket()是将异步模式的套接字再改回阻塞模式，但之前要调用WSAAsynSelect()取消所有的异步事件,WSAAsynSelect(s,hWnd,0,0);

（3）启动与终止 启动函数WSAStartup()建立于Winsock DLL 的连接， 终止函数WSAClearup()终止该DLL，这两个函数必须成对使用。

（4）出错处理 Winsock为了与以后的多线程环境相兼容，提供了两个出错处理函数来获取和设置当前线程的最近错误号，即WSAGetLastError()和WSASetLastError();