 近日，由于程序设计需要，我对WincowsCE 的内存布局进行了研究，由于发现国内在这方面的文档资料较少，于是在研究告一段落之际，形成这篇示例文档，以望抛砖引玉，得到别的高手的指正。 一、程序实现的先决条件 由于Windows系统的窗体消息总是投递至一个特定进程的指定窗体消息函数中。于是在本地进程(自己的应用程序)中取得属于其它进程的窗体的消息必须实现以下两个部分： 1、将需要挂接窗体的代码放到目标进程的地址空间中去。 2、执行这一段代码，并获得目标进程窗体的消息。 这两步看起来很简单，但在实现过程中就比较困难。由于Windows CE作为嵌入式移动设备操作系统，与Windows 98/2000/XP等桌面操作系统在内核的设计理念以及API的支持上有极大的区别。这就直接导致了常规的桌面系统利用全局鼠标钩子注入/远程线程注入等方法在CE中完全得不通。不过可喜的是，微软在开发工具中提供的remotexxx等远程调试程序使我清楚这个目标并不是不可能的任务，微软既然可以做到，那就是说在CE的内部一定有一套完整的跨进程内存访问/代码注入的机制。 二、程序实现的基本原理 经过两天的google 搜索，在网上我发现了一个没有在微软文档中声明的有趣的API函数：PerformCallBack4，传说中这个函数可以在自己的应用程序中执行指定的进程中的一个函数，So Cool! 这好象正是我所需要的东西。虽然网上也传闻这个函数在wm5不受支持，其实经过实践这个传闻只是谣传而已!

PerformCallBack4函数的定义： [DllImport("coredll.dll")] public static extern uint PerformCallBack4(ref CallBackInfo CallBackInfo, IntPtr ni_pVoid1,IntPtr ni_pVoid2,IntPtr ni_pVoid3). 其中函数的参数CallBackInfo结构定义:
[StructLayout(LayoutKind.Sequential)] public struct CallBackInfo { public IntPtr hProc. //远程的目标进程本文来源:百考试题网 public IntPtr pfn. //指向远程目标进程的函数地址的指针 public IntPtr pvArg0. //函数的需要的第一个参数 }//end struct
而PerformCallback4的 ni_pVoid1、ni_pVoid2、ni_pVoid3为传递到远程目标进程执行函数的其它三个参数。 至于将代码放到目标进程的内存空间，我们可以利用CE设计上的一个特性：1、为了节约内存使用，CE将所有程序调用的动态链接库(DLL)都映射到同一个内存地址中。 2、CE的内存布局中划分有一个slot0的内存位置，这个内存位置是由正在执行的进程所占有的，每一个特定的时间片，只能有一个进程可以占有这个内存空间。在进程要求执行时，系统并不直接执行进程所处内存位置的代码，而是将该进程的执行代码复制到slot0的内存位置中产生一个副本执行。也就是说进程在执行时内存将会有进程执行代码的两个完全一样的版本：存在于slot0中正在执行的进程代码和进程本身所处的内存中的代码。 在这个特性下，可以得到结论：如果进程A通过LoadLibrary函数装载Test.dll，而进程B也通过LoadLibrary函数装载同一个Test.dll，这个Test.dll的所有函数在进程A和进程B中执行时，相对于slot0中的进程执行代码都会得到同一地址。 3、在CE中，系统在内存中划分出33个slot，slot0保留给正在执行的进程，然后在进程启动时将所有的代码放到

除slot0以外的一个 slot中(这就是臭名昭著的CE系统中内存最多只能有不多于32个程序执行的限制的来由)。在进程执行时，每个应用程序的内存访问默认只能访问 slot0内存空间中的地址以及进程所处的slot内存空间的地址。 但为使设备驱动程序可以访问到它们所需的其它应用程序数据，CE提供了两个函数以打破这个限制，SetKmode和SetProcPermission，SetKmode 函数告诉系统，当前运行的进程是否需要在内核模式中执行.SetProcPermission函数可以接受一个位掩码，每一位代码一个slot的访问控制，1代表可以访问该slot的内存内容。0表示不能访问该slot的内存内容。这两个函数在msdn中有帮助文档，可参阅msdn的文档说明。 