用C++Builder为接口开发设计应用程序

王小林,刘宏申
(安徽工业大学 计算机学院,安徽 马鞍山 243002)

摘 要:介绍了驱动程序的模型、可执行文件的执行原理及格式,分析了Borland公司和Microsoft公司在obj文件、lib文件格式方面的不同,得出二者不能通用的原因,最后介绍如何用C++Builder为DLL生成自己需要的lib文件。文中内容对C++Builder用户、Delphi用户在接口编程开发上有一定的指导作用。

关键词:C++Builder;接口;lib文件;应用程序
中图分类号:TP311.1 **文献标识码**:A **文章编号**:1673-629X(2006)06-0015-02

Development of Application Programs for Hardware Interface with C++Builder

WANG Xiao-lin, LIU Hong-shen
(School of Computer Science, Anhui University of Technology, Maanshan 243002, China)

Abstract: The model of driver program in Windows is introduced firstly. The format and executale principle of executable file are presented and the differences of .obj files and .lib files between Borland and Microsoft are analyzed. The method of generating .lib file for DLL file is presented. The paper is guidance of user of C++Builder and Delphi.

Key words: C++Builder; hardware interface; .lib file; application program

0 引 言

C++Builder作为C++类的一个开发工具,尽管没有VC那样博大精深,但因其易学易用、开发效率高而深得用户的青睐。大多数C++Builder用户都有这种看法:即在一般应用程序的开发上它不逊于任何其它开发工具,但在开发基于某个硬件板卡的应用程序时有时就不方便。主要原因不是C++Builder本身的原因,而是硬件板卡厂商在提供的开发文档中有关资料较少和对C++Builder有些功能不了解所致。多数硬件板卡厂商除了提供驱动程序的安装程序、供用户编程使用的函数接口API(一般以DLL文件形式存在)外,还提供了使用这些DLL文件的lib文件和示范程序,遗憾的是多数厂商只提供VC的下lib文件,在C++Builder中不能使用。正是这个原因导致较多C++Builder用户有上面的无奈。实际上C++Builder本身提供了解决办法,这就是文中的主要内容。

1 NT平台下驱动程序模型(WDM)

计算机的板卡实际上就是计算机系统中外部接口设备,它主要是起数据格式转换、数据缓冲等作用。一个板卡生产商要使其设计的板卡在NT平台下可用,必须为此板卡设计在NT平台下驱动程序,并为板卡用户提供编程用的API函数,这些API函数一般是以动态连接库形式存在。用户的应用程序、板卡的DLL文件和板卡驱动程序的关系如下(见图1):

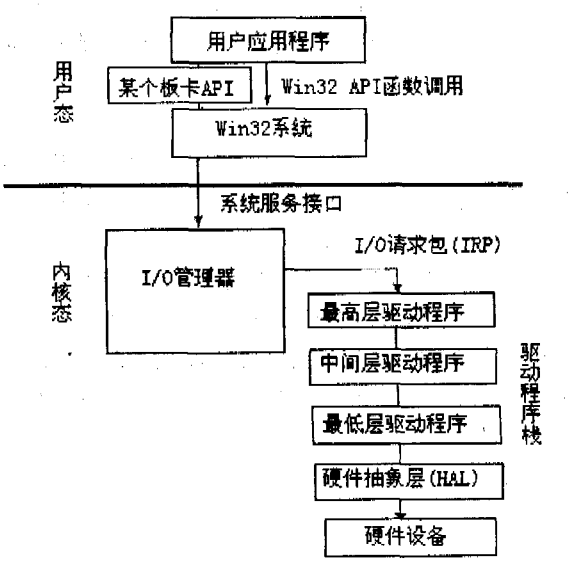

图1 设备驱动程序的调用

板卡厂商按照Windows驱动程序模型(WDM)[1]规范设计的驱动程序位于驱动程序栈中某个层次,由系统的I/O管理器管理。用户的应用程序对设备的操作实际上是调用板卡厂商提供的API和调用Win32系统API函

收稿日期:2005-09-09
基金项目:安徽省教育厅自然科学基金资助项目(2005KJ071)
作者简介:王小林(1964-),男,安徽安庆人,副教授,研究方向为软件工程和人工智能。

数,最终归为由 Win32 系统通过系统服务接口,将调用信息传递给 I/O 管理器,接着由 I/O 管理器根据这个请求构造一个 I/O 请求包 IRP(I/O Request Packet),一般会直接把 IRP 传递给设备的驱动程序,由驱动程序中的相应例程处理该 IRP。由于 WDM 使用层次式的驱动程序概念,所以必要时驱动程序还要把这个 IRP 传递给驱动程序设备栈中的下一级驱动程序,当然如果当前的驱动程序不适合处理该 IRP,也可以直接把 HIP 向下层传递。在完成了 IRP 的处理后,I/O 管理器把数据和结果返回给 Win32 和用户应用程序。上述 Win32 系统和板卡的 API 都是以动态连接库文件(DLL)形式存在, Win32 系统包括 user32.dll 和 Kernal32.dll。其中板卡的 API 可看作 Win32 系统 API 的扩展,它是在开发驱动程序时生成的。

2 应用程序的格式及产生

众所周知,现在的计算机都是冯诺依曼体系的,它能执行的程序是存放在内存中有序的指令序列。任何一个程序不管是用何工具、在何环境中生成的,全部装入或部分装入内存执行时都必须是在时间上顺序已定的指令序列。这个在执行时间上有序指令序列可看成连续逻辑地址空间,它实际上就是虚拟地址空间。每个程序的逻辑地址空间(虚拟地址空间)是独立的、连续的、从 0 开始编址的。一个应用程序就是一个可执行文件,一个可执行文件如果扩展名为.exe 时,它还不是一个计算机机器指令在连续逻辑地址空间的有序序列,它仅仅是包含了一些模块和将这些模块重定位或"组装"成连续逻辑地址空间的有序序列的信息的文件。它之所以能执行是在执行时由系统根据它提供的组装信息将它各模块组装成一个有序序列。不同操作系统环境支持的可执行文件格式可能是不同的,在 Windows 平台下可执行文件的格式是 PE 格式。因此在 Windows 平台下不管什么编程工具生成的可执行程序都必须是这种格式[2]。

在 Windows 下,无论在什么编程工具中可执行文件的产生都是先将高级语言编写的程序编译成.obj 文件,然后由连接程序将生成的.obj 文件、库文件(.lib 文件)和 DLL 文件连接生成具有 PE 格式的可执行文件。不同的编程工具可以有自己格式的.obj 文件、库文件(.lib 文件)和相应的连接程序,它只要保证最后生成的文件满足 PE 格式即可。Borland 公司和 Microsoft 公司的.obj 文件、库文件(.lib 文件)采用的格式是不同的[3~5],前者采用 OMF 格式,而后者采用 COFF 格式。因此二者的.obj 文件、库文件(.lib 文件)是不能通用的,VC 中使用的 lib 文件在 C++Builder 中不能使用。

3 DLL 文件及连接

DLL 文件是 Windows 下出现的一种新的模块公用机制。使用普通的函数库建立可执行文件时是将库中代码拷贝到可执行文件中;当使用 DLL 建立可执行文件时,不将其中代码拷贝到可执行文件中,只是在程序中记录了函数的入口点。在应用程序运行时动态地装载 DLL 并被映射到进程的地址空间中。不管多少程序使用了该 DLL,内存中都只有该 DLL 的一个副本。在 DLL 中一般有两种函数:内部函数和导出函数(export),其中导出函数就是供使用 DLL 的应用程序调用的,DLL 文件中包含一个导出函数表,它包含了函数的符号化名字和函数在 DLL 内的地址。

在应用程序编程时,如需要调用 DLL 的导出函数,必须对调用的 DLL 的导出函数声明。由于 DLL 连接机理与普通函数库不同,因此在应用程序中对调用的 DLL 的导出函数声明与普通函数的声明不同,在函数声明中要加一些修辞符。应用程序运行前或运行中要加载 DLL 到进程地址空间,称这为连接。DLL 连接有两种方式:即显式连接和隐式连接。显式连接就是 DLL 的装载由应用程序自己决定加载的时机,装载是通过调用函数 LoadLibrary、GetProcAddress 来实现的。隐式连接需要为该 DLL 文件生成一个 lib 文件,在开发应用程序时只要将与该 DLL 相对应的 lib 文件和头文件加入工程中即可,每当应用程序运行时都要检查它需要的 DLL 文件是否已装入内存,如果未装入则装入。比较二种方式,显示连接可以不需要声明函数的头文件,也不需要 lib 文件,但在程序中需要定义与调用函数原型一致的函数指针;隐式连接只要将头文件和 lib 文件加入即可,不要自己声明函数原型并加载,因此使用方便一些,尤其在导出函数很多时。文中主要讨论隐式连接下的接口编程问题。

4 在 C++Builder 中 DLL 文件的 lib 文件的产生

正如上面分析的,VC 和 C++Builder 中 lib 文件采用的规范不同,VC 采用 COFF 格式而 C++Builder 采用 OMF 格式,二者不能混用。而很多的板卡生产厂家在提供软件时只提供 DLL 文件、VC 下 lib 文件和使用示范程序,因此有些 C++Builder 用户有此看法:C++Builder 开发程序效率很高,但用来在板卡的二次开发上存在厂商不支持的说法。实际上 C++Builder 提供了解决的工具,只不过大家所不知而已。大家对 C++Builder 的可视化的集成开发环境中的功能应该比较了解,介绍的文献书籍比较多,而对它们提供的命令行工具知之甚少。C++Builder 提供的这些命令行工具都是运行在 DOS 方式下,这些程序应该在安装的 C++Builder 的目录路径中的 \bin 目录下。C++Builder 中有一个命令行的工具 implib.exe,它能对给定的 DLL 文件生成 C++Builder 可用的 lib 文件,具体用法如下:

implib [参数] lib 文件名[.lib]DLL 文件 1 DLL 文件 2…

其中"lib 文件名"是要生成的 lib 文件名,"DLL 文件 1"、"DLL 文件 2"等是欲对其生成 lib 文件的 DLL 文件。

(下转第 19 页)

是权限的集合,一个权限状态机用来动态地控制角色的权限集合。

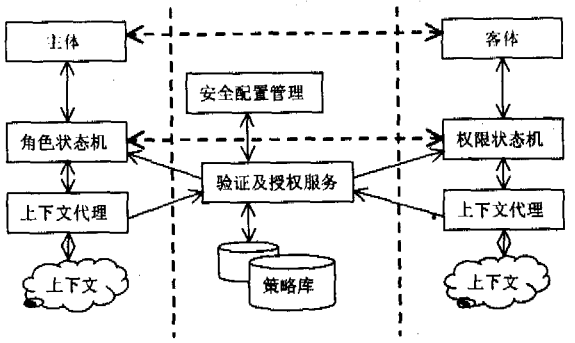

图 1　动态网格访问控制模型

上下文代理:由 AAS 分配到指定的主体和客体处,它们使用中间件服务来监测上下文并产生事件触发状态机的状态迁移。

角色状态机:用于维护主体的活动角色子集,它受上下文代理驱动。

权限状态机:用于维护客体角色的活动权限子集,它受上下文代理驱动。

基于该模型的网格访问控制基本流程图如图 2 所示。

用户首先登录,当通过登录验证后,验证及授权服务授予用户角色集合并启动相关的角色状态机,委派上下文代理到用户宿主环境,上下文代理控制角色状态机动态调整活动角色。资源和服务一旦被允许进入网格提供服务,则对应的角色的权限状态机启动,一个角色可能和多个资源相关。权限状态机根据资源和服务以及其他情形对角色的权限集进行动态调整。对某一时刻的资源访问意味着在此刻的上下文下,用户属于一定的角色,并且该角色在一定的上下文下对资源进行某些操作。

4　结束语

介绍了基于角色的访问控制的标准 RBAC,着重于解决网格计算访问控制问题,提出了基于标准角色访问控制的扩展方法来解决网格计算访问控制的动态控制、自主上下文感知等问题,并且给出了网格访问控制模型的描述。不难看出扩展的 RBAC 方法的主要有内容简单、多粒度控制、易于实现、易于扩展等优点。基于扩展的 RBAC 给出了一个动态模型,该模型强调了上下文下对用户的角色进行调整,比较适合网格环境下的资源访问控制。未来工作是结合对该模型的实际应用做进一步研究。

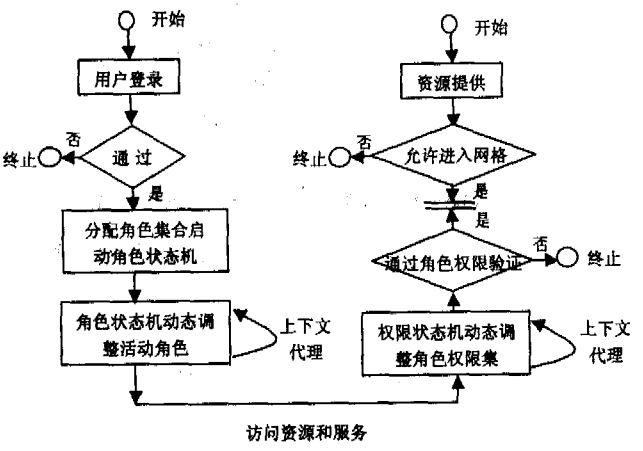

图 2　网格访问控制基本流程

参考文献:

[1] 宁　葵,严　毅,李陶深. 一种基于角色访问控制的数据库安全模型[J]. 微机发展,2005,15(10):8-10.

[2] 赵　亮. 访问控制研究综述[J]. 计算机工程,2004,30(2):1-2.

[3] 徐志伟,冯百明,李　伟. 网格计算技术[M]. 北京:电子工业出版社,2004.

[4] 张　纲,李晓林,游赣梅,等. 基于角色的信息网格访问控制的研究[J]. 计算机研究与发展,2002,39(8):952-956.

[5] 陈　华. 网格的访问控制模型[J]. 微机发展,2004,14(8):27-29.

(上接第 16 页)

在实际编程时,只要将要连接的 DLL 文件、相应的头文件和用上述方法生成的 lib 文件拷贝当前工程项目的目录中即可。

5　结束语

由于 Borland 公司和 Microsoft 公司在 obj 文件、lib 文件格式方面的不同,因此二者的 obj 文件、lib 文件不能混用。C++Builder 在开发硬件接口方面的应用程序时是提供了相应的工具来解决 lib 文件格式不兼容的问题,即 C++Builder 的命令行工具 implib。Implib 能由动态连接文件(DLL 文件)生成相应的 lib 文件。

参考文献:

[1] Cant C. Windows WDM 设备驱动程序开发指南[M]. 北京:机械工业出版社,2000.

[2] Richter J. Windows 核心编程[M]. 北京:机械工业出版社,2000.

[3] Pietrek M. Under the hood: Link-time Code Generation[J/OL]. http://msdn.microsoft.com/msdnmag/issues/02/05/Hood/. 1996.

[4] Pietrek M. Inside Windows: An In-Depth Look into the Win32 Portable Executable File Format[J/OL]. http://msdn.microsoft.com/msdnmag/issues/02/02/PE/default.aspx. 2002.

[5] Pietrek M. Inside Windows: An In-Depth Look into the Win32 Portable Executable File Format (Part2)[J/OL]. http://msdn.microsoft.com/msdnmag/issues/02/03/PE2/default.aspx. 2002.