

<<Windows核心编程>>

图书基本信息

书名：<<Windows核心编程>>

13位ISBN编号：9787111237914

10位ISBN编号：7111237919

出版时间：2008-5

出版时间：机械工业出版社

作者：理查德

页数：728

译者：黄陇,李虎

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

<<Windows核心编程>>

内容概要

本书是讲解Windows操作系统内部机制的一本专著。作者从基本概念入手，全面系统地介绍了Windows底层实现机制、Windows应用程序的基本构件（包括进程、线程、内存管理、动态链接库、线程本地存储和Unicode）以及各类Windows API等，并列举了大量应用程序示例，精辟地分析了Windows编程的各个难点和要点，为掌握Windows编程技巧提供了一条有效的捷径。

本书适合Windows编程人员参考。

作者简介

作者：(美)理查德 译者：黄隴 李虎 Jeffrey Richter，是一位在全球享有盛誉的技术作家，尤其在Windows/.NET领域有着杰出的贡献。

他的第一本Windows著作《Windows 95：A Developer'S Guide》大获好评，从而声名远扬。

之后，他又推出了经典著作《Windows高级编程指南》和《Windows核心编程》。

如今这两本书早已成为Windows程序设计领域的经典之作，培育了几代软件开发设计人员。

Jeffrey是Wintellect公司的创始人之一，也是MSDN杂志.NET专栏的特邀编辑。

他对Windows思想的领悟、对Windows细节的熟稔，是其他任何作家难以企及的。

他是Windows技术作家中当之无愧的一面旗帜。

书籍目录

译者序前言作译者简介第一部分 程序员必读第1章 错误处理1.1 自定义错误处理的实现1.2 错误显示例程第2章 Unicode2.1 字符集2.1.1 单字节和双字节字符集2.1.2 Unicode：宽字节字符集2.2 为何需要Unicode2.3 Windows 2000和Unicode2.4 Windows 98和Unicode2.5 Windows CE和Unicode2.6 评论2.7 关于COM2.8 如何编写Unicode源代码2.8.1 C运行库的Unicode支持2.8.2 Windows定义的Unicode数据类型2.8.3 Windows系统中的Unicode函数和ANSI函数2.8.4 Windows字符串函数2.9 让应用程序符合ANSI和Unicode规范2.9.1 Windows字符串函数2.9.2 资源2.9.3 确定文本是ANSI型还是Unicode型2.9.4 在Unicode和ANSI间转换字符串第3章 内核对象3.1 内核对象的概念3.1.1 使用计数3.1.2 安全性3.2 内核对象句柄表3.2.1 创建内核对象3.2.2 关闭内核对象3.3 进程间内核对象的共享3.3.1 对象句柄的继承性3.3.2 改变句柄标志3.3.3 命名对象3.3.4 终端服务器命名空间3.3.5 复制对象句柄第二部分 完成编程任务第4章 进程4.1 编写第一个Windows应用程序4.1.1 进程的实例句柄4.1.2 进程的前一个实例句柄4.1.3 进程的命令行4.1.4 进程的环境变量4.1.5 亲缘性4.1.6 进程的故障模式4.1.7 当前驱动器和目录4.1.8 当前目录4.1.9 系统版本4.2 CreateProcess函数4.2.1 pszApplicationName和pszCommandLine4.2.2 psaProcess、psaThread和bInheritHandles4.2.3 fdwCreate4.2.4 pvEnvironment4.2.5 pszCurDir4.2.6 psiStartInfo4.2.7 ppiProcInfo4.3 进程的终止4.3.1 主线程的入口函数返回4.3.2 ExitProcess函数4.3.3 TerminateProcess函数4.3.4 进程中所有线程的运行终止4.3.5 进程的运行终止4.4 子进程4.5 枚举系统中运行的进程第5章 作业5.1 对作业进程的限制5.2 把进程放入作业5.3 终止作业中所有进程的运行5.4 查询作业统计信息5.5 作业通知信息5.6 JobLab示例应用程序第6章 线程的基本知识6.1 创建线程的时机6.2 何时不能创建线程6.3 编写第一个线程函数6.4 CreateThread函数6.4.1 psa6.4.2 cbStack6.4.3 pfnStartAddr和pvParam6.4.4 fdwCreate6.4.5 pdwThreadId6.5 终止线程6.5.1 线程函数返回6.5.2 ExitThread函数6.5.3 TerminateThread函数6.5.4 在进程终止运行时终止线程6.5.5 线程终止运行时发生的操作6.6 线程的一些内部细节6.7 对于C/C++运行时库的考虑6.7.1 Oops-错误地调用了CreateThread6.7.2 不该调用的C/C++运行时库函数6.8 线程的身份标识第7章 线程的调度、优先级和亲缘性7.1 挂起和恢复线程的运行7.2 进程的挂起和唤醒7.3 睡眠7.4 线程切换7.5 线程的运行时间7.6 上下文环境切换7.7 线程优先级7.8 优先级的抽象说明7.9 编程优先级7.9.1 动态提高线程的优先级等级7.9.2 为前台进程调整调度程序7.9.3 Scheduling Lab示例应用程序7.10 亲缘性第8章 用户模式下的线程同步8.1 原子访问：互锁函数族8.2 高速缓存行8.3 高级线程同步8.4 临界区8.4.1 临界区准确的描述8.4.2 临界区与循环锁8.4.3 临界区与错误处理8.4.4 有用的提示和技巧第9章 线程与内核对象的同步9.1 等待函数9.2 成功等待的副作用9.3 事件内核对象9.4 等待定时器内核对象9.4.1 用等待定时器给APC项排队9.4.2 定时器的松散特性9.5 信号量内核对象9.6 互斥内核对象9.6.1 释放问题9.6.2 互斥对象与临界区的比较9.6.3 Queue应用程序示例9.7 线程同步对象表9.8 其他线程同步函数9.8.1 异步设备I/O9.8.2 WaitForInputIdle9.8.3 MsgWaitForMultipleObjects(Ex)9.8.4 WaitForDebugEvent9.8.5 SignalObjectAndWait第10章 线程同步工具包10.1 临界区的实现：Optex10.2 创建线程安全的数据类型和反信号量10.3 单写入多读出程序的保护10.4 WaitForMultipleExpressions函数的实现第11章 线程池11.1 场景1：异步调用函数11.2 场景2：按规定的时间间隔调用函数11.3 场景3：在某个内核对象变为已通知状态时调用函数11.4 场景4：异步I/O请求运行完成时调用函数第12章 线程12.1 使用线程12.2 Counter示例应用程序第三部分 内存管理第13章 Windows内存结构13.1 进程的虚拟地址空间13.2 虚拟地址空间分区13.2.1 无效断点分配分区（适于Windows 2000和Windows 98）13.2.2 MS-DOS/16位Windows应用程序兼容分区（仅适于Windows 98）13.2.3 用户模式分区（适用Windows 2000和Windows 98）13.2.4 64KB禁止进入分区（仅适用于Windows 2000）13.2.5 共享的MMF分区（仅适用于Windows 98）13.2.6 内核模式分区（使用于Windows 2000和Windows 98）13.3 地址空间区域13.4 在地址空间区域中提交物理存储器13.5 物理存储器和页面文件13.6 保护属性13.6.1 Copy-On-Write访问13.6.2 特殊访问保护属性标志13.7 综合使用所有元素13.7.1 区域的内部详情13.7.2 Windows 98上地址空间的差异13.8 数据对齐的重要性第14章 虚拟内存14.1 系统信息14.2 虚拟内存的状态14.3 确定地址空间状态14.3.1 VMQuery函数14.3.2 虚拟内存表应用程序示例第15章 应用程序中虚拟内存的使用15.1 地址空间中保留区域15.2 在保留区域中提交存储器15.3 同时进行保留区域并提交内存15.4 何时提交物理存储器15.5 物理存储器的回收和地址空间区域的释放15.5.1 何时回收物理存储器15.5.2 虚拟内存分配示例应用程序15.6 改变保护属性15.7 清除

物理存储器内容15.8 地址窗口扩展（仅使用于Windows 2000）第16章 线程栈16.1 Windows 98下的线程栈16.2 C/C++运行时库中的栈检测函数16.3 Summation示例应用程序第17章 内存映射文件17.1 内存映射的可执行文件和DLL文件17.1.1 可执行文件或DLL的多个实例之间无法共享的静态数据17.1.2 在可执行文件或DLL的多个实例之间共享静态数据17.1.3 AppInst示例应用程序17.2 内存映射数据文件17.2.1 方法1：一个文件，一个缓存17.2.2 方法2：两个文件，一个缓存17.2.3 方法3：一个文件，两个缓存17.2.4 方法4：一个文件，零个缓存17.3 使用内存映射文件17.3.1 步骤1：创建或打开文件内核对象17.3.2 步骤2：创建文件映射内核对象17.3.3 步骤3：将文件数据映射到进程地址空间17.3.4 步骤4：进程地址空间中撤销文件数据的映像17.3.5 步骤5和步骤6：关闭文件映射对象和文件对象17.3.6 文件倒序示例应用程序17.4 使用内存映射文件处理大文件17.5 内存映射文件的一致性17.6 设定内存映射文件的基地址17.7 实现内存映射文件的具体细节17.8 使用内存映射文件在进程之间实现数据共享17.9 受页面文件支持的内存映射文件17.10 稀疏提交的内存映射文件第18章 堆18.1 进程的默认堆18.2 创建辅助堆的原因18.2.1 保护组件18.2.2 更有效地管理内存18.2.3 进行本地访问18.2.4 减少线程同步开销18.2.5 快速释放18.3 创建辅助堆的方法18.3.1 分配堆中的内存块18.3.2 改变内存块的大小18.3.3 获取内存块的大小18.3.4 释放内存块18.3.5 销毁堆18.3.6 用C++程序使用堆18.4 其他堆函数第四部分 动态链接库第19章 DLL基础19.1 DLL与进程的地址空间19.2 DLL的总体运行情况19.3 创建DLL模块19.3.1 导出的真正含义19.3.2 使用非Visual C++工具创建DLL19.4 创建可执行模块19.5 运行可执行模块第20章 DLL高级技术20.1 显式加载DLL模块和符号链接20.1.1 显式加载DLL模块20.1.2 显式卸载DLL模块20.1.3 显式链接到导出符号20.2 DLL的入口函数20.2.1 DLL_PROCESS_ATTACH通知20.2.2 DLL_PROCESS_DETACH通知20.2.3 DLL_THREAD_ATTACH通知20.2.4 DLL_THREAD_DETACH通知20.2.5 顺序调用DllMain20.2.6 DllMain和C/C++运行时库20.3 延迟加载DLL20.4 函数转发器20.5 已知的DLL20.6 DLL重定向20.7 模块的基址重置20.8 绑定模块第21章 线程本地存储21.1 动态TLS21.2 静态TLS第22章 DLL注入以及API挂接22.1 DLL注入：一个例子22.2 使用注册表注入DLL22.3 使用Windows钩子注入DLL22.4 使用远程线程注入DLL22.4.1 Inject Library示例应用程序22.4.2 Image Walk DLL22.5 使用特洛伊DLL注入DLL22.6 将DLL作为调试程序注入22.7 在Windows 98平台上使用内存映射文件注入代码22.8 使用CreateProcess来注入代码22.9 API挂接：一个例子22.9.1 通过覆写代码实现API挂接22.9.2 通过操作模块的导入部分来实现API挂接22.9.3 LastMsgBoxInfo示例应用程序第五部分 结构化异常处理第23章 终止处理例程23.1 Funcenstein123.2 Funcenstein223.3 Funcenstein323.4 Funcfurter123.5 小测验：FuncadoodleDoo23.6 Funcenstein423.7 Funcarama123.8 Funcarama223.9 Funcarama323.10 Funcarama4:最终的边界23.11 有关finally块的说明23.12 Funcfurter223.13 SEH终止示例应用程序第24章 异常处理程序和软件异常24.1 通过例子理解异常过滤器和异常处理程序24.1.1 Funcmeister124.1.2 Funcmeister224.2 EXCEPTION_EXECUTE_HANDLER24.2.1 一些有用的例子24.2.2 全局展开24.2.3 暂停全局展开24.3 EXCEPTION_CONTINUE_EXECUTION24.4 EXCEPTION_CONTINUE_SEARCH24.5 GetExceptionCode24.5.1 与内存相关的异常24.5.2 与异常相关的异常24.5.3 与调试相关的异常24.5.4 与整数相关的异常24.5.5 与浮点数相关的异常24.6 GetExceptionInformation24.7 软件异常第25章 未处理异常和C++异常25.1 即时调试25.2 关闭异常消息框25.2.1 强制进程终止运行25.2.2 包装一个线程函数25.2.3 包装所有的线程函数25.2.4 自动调用调试器25.3 自己调用UnhandledExceptionFilter25.4 UnhandledExceptionFilter函数的内部细节25.5 异常和调试程序25.6 C++异常与结构化异常比较第六部分 窗口第26章 窗口消息26.1 线程的消息队列26.2 将消息投送到一个线程的消息队列中26.3 向窗口发送消息26.4 唤醒一个线程26.4.1 队列状态标志26.4.2 从线程队列中提取消息的算法26.4.3 使用内核对象或者队列状态标志来唤醒一个线程26.5 使用消息发送数据26.6 Windows处理ANSI/Unicode字符和字符串的方法第27章 硬件输入模型与本地输入状态27.1 原始输入线程27.2 本地输入状态27.2.1 键盘输入和焦点27.2.2 鼠标光标管理27.3 将虚拟输入队列和本地输入状态相关联27.3.1 LISLab示例应用程序27.3.2 LISWatch示例应用程序附录附录A 构建环境附录B 消息解析器、子控件宏以及API宏

章节摘录

第一部分 程序员必读第1章 错误处理在开始学习Windows提供的一些必要特性之前，有必要了解各种Windows函数是如何进行错误处理的。

当调用一个Windows函数时，系统首先验证传递给该函数的参数是否有效，然后开始执行该函数的任务。

如果函数接收到一个无效的参数，或者由于其他原因导致该函数不能执行，则通过某种返回值来指示函数执行失败。

表1-1给出了大多数Windows函数用到的返回值的数据类型。

当一个Windows函数返回一个错误代码时，了解该函数为何执行失败往往非常有用。

微软公司已经编译了一系列可能的错误代码，而且为每个错误代码赋予了一个32位的数字。

在系统内部，当一个Windows函数检测到一个错误时，它常常使用一种称为线程本地存储（thread-local storage）的机制来将合适的错误代码号和被调用的线程联系起来（线程一本地存储机制在第21章中讨论）。

这样就允许线程之间互相独立地运行，而不会影响彼此的错误代码。

当函数返回其值后，该返回值会指示发生了一个错误。

可调用GetLastError函数查看错误详情。

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>