

<<Windows核心编程>>

图书基本信息

书名：<<Windows核心编程>>

13位ISBN编号：9787115182128

10位ISBN编号：7115182124

出版时间：2008年8月

出版单位：人民邮电出版社

作者：Jeffrey Richter, Christophe Nasarre

页数：820

字数：1013000

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

书是Windows程序设计领域的名著，涵盖了Windows的最新版本Vista 以及Windows XP的最新内容。书中全面深入地介绍了Windows的各种基本要素，如进程、线程池、虚拟内存、DLL、设备I/O和SEH等，并列举了大量应用程序，精辟地分析了要素的使用方法。

本书适于各层次Windows编程人员阅读。

作者简介

Jeffrey Richter全球享有盛誉的微软技术专家，著名技术咨询和培训公司Wintellect创始人之一，MSDN Magazine杂志特邀编辑。多年来，他担任微软各开发团队顾问，参与了微软的许多关键产品包括各版本Windows、Visual Studio、Microsoft Office和.NET框架的设计和编程。他撰写

<<Windows核心编程>>

书籍目录

| | | | | | |
|---|--|---|--|---|---------------------------------------|
| Part I | Required Reading | 1 | Error Handling | Defining Your Own Error Codes | The ErrorShow |
| Sample Application | | 2 | Working with Characters and Strings | Character Encodings | |
| | ANSI and Unicode Character and String Data Types | | | Unicode and ANSI Functions in Windows | |
| | Unicode and ANSI Functions in the C Run-Time Library | | | Secure String Functions in the C | |
| Run-Time Library | | | Introducing the New Secure String Functions | How to Get More | |
| Control When Performing String Operations | | | Windows String Functions | Why You Should | |
| Use Unicode | | | How We Recommend Working with Characters and Strings | Translating Strings | |
| Between Unicode and ANSI | | | Exporting ANSI and Unicode DLL Functions | Determining | |
| If Text Is ANSI or Unicode | | 3 | Kernel Objects | What Is a Kernel Object? | Usage |
| Counting | Security | | A Process ' Kernel Object Handle Table | Creating a Kernel | |
| Object | Closing a Kernel Object | | Sharing Kernel Objects Across Process Boundaries | | |
| | Using Object Handle Inheritance | | Naming Objects | Duplicating Object Handles | |
| Part II | Getting Work Done | 4 | Processes | Writing Your First Windows Application | A |
| Process Instance Handle | | | The CreateProcess Function | pszApplicationName and | |
| pszCommandLine | | | Terminating a Process | The Primary Thread ' s Entry-Point Function | |
| Returns | The ExitProcess Function | | | The TerminateProcess Function | When All |
| the Threads in the Process Die | | | When a Process Terminates | Child Processes | |
| Running Detached Child Processes | | | When Administrator Runs as a Standard User | | |
| Elevating a Process Automatically | | | Elevating a Process by Hand | What Is the Current | |
| Privileges Context? | Enumerating the Processes Running in the System | | | 5 Jobs | |
| Placing Restrictions on a Job ' s Processes | Placing a Process in a Job | | | Terminating All | |
| Processes in a Job Querying Job Statistics | Job Notifications | | | The Job Lab Sample Application | |
| 6 Thread Basics | When to Create a Thread | | | When Not to Create a Thread | |
| Writing Your First Thread Function | The CreateThread Function | | | psa | |
| cbStackSize | pfnStartAddr and pvParam | | | dwCreateFlags | pdwThreadId |
| Terminating a Thread | The Thread Function Returns | | | The ExitThread Function | |
| The TerminateThread Function | When a Process Terminates | | | When a Thread | |
| Terminates | Some Thread Internals | | | C/C++ Run-Time Library Considerations | |
| Oops—I Called CreateThread Instead of _beginthreadex by Mistake | | | | C/C++ Run-Time Library | |
| Functions That You Should Never Call | Gaining a Sense of One ' s Own Identity | | | Converting | |
| a Pseudohandle to a Real Handle | 7 Thread Scheduling, Priorities, and Affinities | | | Suspending and | |
| Resuming a Thread | Suspending and Resuming a Process | | | Sleeping | Switching to |
| Another Thread | Switching to Another Thread on a Hyper-Threaded CPU | | | | A Thread ' s |
| Execution Times | Putting the CONTEXT in Context | | | Thread Priorities | An Abstract |
| View of Priorities | Programming Priorities | | | Dynamically Boosting Thread Priority Levels | |
| | Tweaking the Scheduler for the Foreground Process | | | Scheduling I/O Request Priorities | |
| | The Scheduling Lab Sample Application | | | Affinities | 8 Thread Synchronization in User Mode |
| | Atomic Access: The Interlocked Family of Functions | | | Cache Lines | Advanced Thread |
| Synchronization | A Technique to Avoid | | | Critical Sections | Critical Sections: The |
| Fine Print | Critical Sections and Spinlocks | | | Critical Sections and Error Handling | |
| | Slim Reader-Writer Locks | | | Condition Variables | The Queue Sample Application |
| | Useful Tips and Techniques | 9 | Thread Synchronization with Kernel Objects | Wait Functions | |
| | Successful Wait Side Effects | | Event Kernel Objects | The Handshake Sample | |
| Application | Waitable Timer Kernel Objects | | | Having Waitable Timers Queue APC Entries | |
| | Timer Loose Ends | | Semaphore Kernel Objects | Mutex Kernel Objects | |

<<Windows核心编程>>

| | | | |
|---|------------|---|--|
| Abandonment Issues | Mutexes vs | Critical Sections | The Queue Sample Application |
| A Handy Thread Synchronization Object Chart | | | Other Thread Synchronization Functions |
| Asynchronous Device I/O | | WaitForInputIdle | MsgWaitForMultipleObjects(Ex) |
| WaitForDebugEvent | | SignalObjectAndWait | Detecting Deadlocks with the |
| Wait Chain Traversal API | 10 | Synchronous and Asynchronous Device I/O | Opening and Closing |
| Devices | | A Detailed Look at CreateFile | Working with File Devices |
| 's Size | | Positioning a File Pointer | Setting the End of a File |
| Synchronous Device I/O | | Flushing Data to the Device | Synchronous I/O Cancellation |
| Basics of Asynchronous Device I/O | | The OVERLAPPED Structure | Asynchronous |
| Device I/O Caveats | | Canceling Queued Device I/O Requests | Receiving Completed I/O |
| Request Notifications | | Signaling a Device Kernel Object | Signaling an Event Kernel Object |
| Alertable I/O | | I/O Completion Ports | 11 The Windows Thread Pool |
| Scenario 1: Call a Function Asynchronously | | Explicitly Controlling a Work Item | The |
| Batch Sample Application | | Scenario 2: Call a Function at a Timed Interval | The Timed |
| Message Box Sample Application | | Scenario 3: Call a Function When a Single Kernel Object Becomes | Signaled |
| Scenario 4: Call a Function When Asynchronous I/O Requests Complete | | | Callback |
| Termination Actions | | Customized Thread Pools | Gracefully Destroying a Thread Pool: |
| Cleanup Groups | 12 | Fibers | Working with Fibers |
| Part III Memory Management | 13 | Windows Memory Architecture | A Process ' Virtual Address |
| Space | | How a Virtual Address Space Is Partitioned | Null-Pointer Assignment Partition |
| User-Mode Partition | | Kernel-Mode Partition | Regions in an Address Space |
| Committing Physical Storage Within a Region | | Physical Storage and the Paging File | Copy-on-Write |
| Physical Storage Not Maintained in the Paging File | | Protection Attributes | Access |
| Special Access Protection Attribute Flags | | Bringing It All Home | Inside the |
| The Importance of Data Alignment | | 14 Exploring Virtual Memory | System |
| The System Information Sample Application | | Virtual Memory Status | Information |
| Memory Management on NUMA Machines | | The Virtual Memory Status Sample Application | |
| Determining the State of an Address Space | | The VMQuery Function | The Virtual |
| Memory Map Sample Application | 15 | Using Virtual Memory in Your Own Applications | |
| Reserving a Region in an Address Space | | Committing Storage in a Reserved Region | Reserving |
| a Region and Committing Storage Simultaneously | | When to Commit Physical Storage | |
| Decommitting Physical Storage and Releasing a Region | | When to Decommit Physical Storage | |
| The Virtual Memory Allocation Sample Application | | Changing Protection Attributes | |
| Resetting the Contents of Physical Storage | | The MemReset Sample Application | Address |
| Windowing Extensions | | The AWE Sample Application | 16 |
| C/C++ Run-Time Library 's Stack-Checking Function | | The Summation Sample Application | 17 |
| Memory-Mapped Files | | Memory-Mapped Executables and DLLs | Static Data Is Not |
| Shared by Multiple Instances of an Executable or a DLL | | Memory-Mapped Data Files | |
| Method 1: One File, One Buffer | | Method 2: Two Files, One Buffer | Method 3: One File, |
| Two Buffers | | Method 4: One File, Zero Buffers | Using Memory-Mapped Files |
| Step 1: Creating or Opening a File Kernel Object | | Step 2: Creating a File-Mapping Kernel Object | |
| Step 3: Mapping the File 's Data into the Process ' Address Space | | Step 4: Unmapping the | |
| File 's Data from the Process ' Address Space | | Steps 5 | and 6: Closing the File-Mapping Object |
| and the File Object | | The File Reverse Sample Application | Processing a Big File Using |
| Memory-Mapped Files | | Memory-Mapped Files and Coherence | Specifying the Base Address of |
| a Memory-Mapped File | | Implementation Details of Memory-Mapped Files | Using |
| Memory-Mapped Files to Share Data Among Processes | | Memory-Mapped Files Backed by the Paging | |

<<Windows核心编程>>

| | | |
|---|---|--|
| File | The Memory-Mapped File Sharing Sample Application | Sparsely Committed |
| Memory-Mapped Files | The Sparse Memory-Mapped File Sample Application | 18 Heaps |
| A Process ' Default Heap | Reasons to Create Additional Heaps | Component Protection |
| More Efficient Memory Management | Local Access | Avoiding Thread |
| Synchronization Overhead | Quick Free | How to Create an Additional Heap |
| Allocating a Block of Memory from a Heap | Changing the Size of a Block | Obtaining the |
| Size of a Block | Freeing a Block | Destroying a Heap |
| Miscellaneous Heap Functions | Part IV Dynamic-Link Libraries | 19 DLL Basics |
| and a Process ' Address Space | The Overall Picture | Building the DLL Module |
| Building the Executable Module | Running the Executable Module | 20 DLL Advanced |
| Techniques | Explicit DLL Module Loading and Symbol Linking | Explicitly Loading the DLL |
| Module | Explicitly Unloading the DLL Module | Explicitly Linking to an Exported Symbol |
| The DLL ' s Entry-Point Function | The DLL_PROCESS_ATTACH Notification | |
| The DLL_PROCESS_DETACH Notification | The DLL_THREAD_ATTACH Notification | |
| The DLL_THREAD_DETACH Notification | Serialized Calls to DllMain | DllMain |
| and the C/C++ Run-Time Library | Delay-Loading a DLL | The DelayLoadApp Sample |
| Application | Function Forwarders | Known DLLs |
| Modules | Binding Modules | 21 Thread-Local Storage |
| Dynamic TLS | Static TLS | 22 DLL Injection and API Hooking |
| Example | Injecting a DLL Using the Registry | Injecting a DLL Using Windows Hooks |
| The Desktop Item Position Saver (DIPS) Utility | | Injecting a DLL Using Remote Threads |
| The Inject Library Sample Application | The Image Walk DLL | Injecting a DLL with a |
| Trojan DLL | Injecting a DLL as a Debugger | Injecting Code with CreateProcess |
| Hooking: An Example | API Hooking by Overwriting Code | API Hooking by |
| Manipulating a Module ' s Import Section | The Last MessageBox Info Sample Application | Part V |
| Structured Exception Handling | 23 Termination Handlers | Understanding Termination Handlers |
| by Example | Funcenstein1 | Funcenstein2 |
| Funcfurter1 | Pop Quiz Time: FuncaDoodleDoo | Funcenstein3 |
| Funcarama2 | Funcarama3 | Funcenstein4 |
| About the finally Block | Funcfurter2 | Funcarama4: The Final Frontier |
| Exception Handlers and Software Exceptions | | Notes |
| Handlers by Example | Funcmeister1 | 24 |
| EXCEPTION_EXECUTE_HANDLER | Some Useful Examples | Global Unwinds |
| Halting Global Unwinds | EXCEPTION_CONTINUE_EXECUTION | Use |
| EXCEPTION_CONTINUE_EXECUTION with Caution | EXCEPTION_CONTINUE_SEARCH | |
| GetExceptionCode | Memory-Related Exceptions | Exception-Related Exceptions |
| Debugging-Related Exceptions | Integer-Related Exceptions | Floating Point |
| - Related Exceptions | GetExceptionInformation | Software Exceptions |
| Exceptions, Vectored Exception Handling, and C++ Exceptions | | 25 Unhandled |
| Function | Action #1: Allowing Write Access to a Resource and Continuing Execution | Inside the UnhandledExceptionHandler |
| | Action #2: Notifying a Debugger of the Unhandled Exception | Action #3: Notifying Your Globally |
| Set Filter Function | Action #4: Notifying a Debugger of the Unhandled Exception (Again) | |
| | Action #5: Silently Terminating the Process | UnhandledExceptionHandler and WER Interactions |
| Just-in-Time Debugging | The Spreadsheet Sample Application | Vectored Exception and |
| Continue Handlers | C++ Exceptions vs Structured Exceptions | Exceptions and the |
| Debugger | 26 Error Reporting and Application Recovery | The Windows Error Reporting Console |
| | Programmatic Windows Error Reporting | Disabling Report Generation and Sending |

<<Windows核心编程>>

Customizing All Problem Reports Within a Process
 Creating a Custom Problem Report: WerReportCreate
 WerReportSetParameter
 Adding Arbitrary Files to the Report: WerReportAddFile
 WerReportSetUIOption
 Problem Report: WerReportCloseHandle
 Automatic Application Restart and Recovery
 for Application Recovery
 Header File
 Windows Definitions and Warning Level 4
 chINRANGE Macro
 x86 Platforms
 chASSERT and chVERIFY Macros
 chSETDLGICONS Macro
 Support XP-Theming of the User Interface with pragma
 Macros, and API Macros
 Index

Creating and Customizing a Problem Report
 Setting Report Parameters:
 Adding a Minidump File to the Report: WerReportAddDump
 Modifying Dialog Box Strings:
 Submitting a Problem Report: WerReportSubmit
 Closing a
 The Customized WER Sample Application
 Automatic Application Restart
 Support
 Part VI Appendixes A The Build Environment
 The CmnHdr.h
 Unicode Build Option
 The pragma message Helper Macro
 The
 DebugBreak Improvement for
 The chMB Macro
 The
 The chHANDLE_DLGMSG Macro
 Forcing the Linker to Look for a (w)WinMain Entry-Point Function
 B Message Crackers, Child Control
 Child Control Macros
 API Macros

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>