Microsoft

5

Fifth Edition

WINDOWS® VIA C/C++

Jeffrey Richter
Christophe Nasarre

Wintellect®
Know how.

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 2008 by Jeffrey Richter and Christophe Nasarre

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Control Number: 2007939306

ISBN: 978-0-7356-6377-0

1 2 3 4 5 6 7 8 9 10 QGT 6 5 4 3 2 1

Printed and bound in the United States of America.

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Microsoft, ActiveX, Developer Studio, Intellisense, Internet Explorer, Microsoft Press, MSDN, MS-DOS, PowerPoint, SQL Server, SuperFetch, Visual Basic, Visual C++, Visual Studio, Win32, Win32s, Windows, Windows Media, Windows NT, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. <possible third-party trademark info>. Other product and company names mentioned herein may be the trademarks of their respective owners.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions Editor: Ben Ryan
Developmental and Project Editor: Lynn Finnel
Editorial Production: Publishing.com
Technical Reviewer: Scott Seely; Technical Review services provided by Content Master, a member of CM Group, Ltd.

Body Part No. X14-25709

Windows® via C/C++

Table of Contents

Dedication

Contents at a Glance

Table of Contents

Acknowledgments

Introduction

- 64-Bit Windows
- Whats New in the Fifth Edition
- Code Samples and System Requirements
- Support for This Book
    - Questions and Comments

Part I: Required Reading

- Chapter 1: Error Handling
    - Defining Your Own Error Codes
    - The ErrorShow Sample Application
- Chapter 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

Pearson

Table of Contents

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

Chapter 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

Chapter 4: Processes

Writing Your First Windows Application

- A Process Instance Handle

The CreateProcess Function

- pszApplicationName and pszCommandLine

Terminating a Process

- The Primary Threads 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

Table of Contents

Elevating a Process Automatically

Elevating a Process by Hand

What Is the Current Privileges Context?

Enumerating the Processes Running in the System

Chapter 5: Jobs

Placing Restrictions on a Jobs Processes

Placing a Process in a Job

Terminating All Processes in a Job

Querying Job Statistics

Job Notifications

The Job Lab Sample Application

Chapter 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

OopsI Called CreateThread Instead of _beginthreadex by Mistake

C/C++ Run-Time Library Functions That You Should Never Call

Pearson

Table of Contents

Gaining a Sense of Ones Own Identity

Converting a Pseudohandle to a Real Handle

Chapter 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 Threads 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

Chapter 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

Pearson

Table of Contents

Useful Tips and Techniques

Chapter 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

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

Chapter 10: Synchronous and Asynchronous Device I/O

Opening and Closing Devices

A Detailed Look at CreateFile

Working with File Devices

Getting a Files Size

Positioning a File Pointer

Setting the End of a File

Performing Synchronous Device I/O

Flushing Data to the Device

Pearson

Table of Contents

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

Chapter 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

Chapter 12: Fibers

Working with Fibers

The Counter Sample Application

Part III: Memory Management

Chapter 13: Windows Memory Architecture

A Process Virtual Address Space

How a Virtual Address Space Is Partitioned

Null-Pointer Assignment Partition

User-Mode Partition

Pearson

Table of Contents

Kernel-Mode Partition

Regions in an Address Space

Committing Physical Storage Within a Region

Physical Storage and the Paging File

Physical Storage Not Maintained in the Paging File

Protection Attributes

Copy-on-Write Access

Special Access Protection Attribute Flags

Bringing It All Home

Inside the Regions

The Importance of Data Alignment

Chapter 14: Exploring Virtual Memory

System Information

The System Information Sample Application

Virtual Memory Status

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

Chapter 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

Pearson

Table of Contents

The MemReset Sample Application

Address Windowing Extensions

The AWE Sample Application

Chapter 16: A Threads Stack

The C/C++ Run-Time Librarys Stack-Checking Function

The Summation Sample Application

Chapter 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 Files Data into the Process Address Space

Step 4: Unmapping the Files 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 File

The Memory-Mapped File Sharing Sample Application

Sparsely Committed Memory-Mapped Files

The Sparse Memory-Mapped File Sample Application

Chapter 18: Heaps

Table of Contents

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
- Using Heaps with C++

Miscellaneous Heap Functions

Part IV: Dynamic-Link Libraries

Chapter 19: DLL Basics

DLLs and a Process Address Space

The Overall Picture

- Building the DLL Module
- Building the Executable Module
- Running the Executable Module

Chapter 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 DLLs Entry-Point Function

- The DLL_PROCESS_ATTACH Notification
- The DLL_PROCESS_DETACH Notification
- The DLL_THREAD_ATTACH Notification

Table of Contents

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

DLL Redirection

Rebasing Modules

Binding Modules

Chapter 21: Thread-Local Storage

Dynamic TLS

Using Dynamic TLS

Static TLS

Chapter 22: DLL Injection and API Hooking

DLL Injection: An 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

API Hooking: An Example

API Hooking by Overwriting Code

API Hooking by Manipulating a Modules Import Section

The Last MessageBox Info Sample Application

Part V: Structured Exception Handling

Table of Contents

Chapter 23: Termination Handlers

Understanding Termination Handlers by Example

Funcenstein1

Funcenstein2

Funcenstein3

Funcfurter1

Pop Quiz Time: FuncaDoodleDoo

Funcenstein4

Funcarama1

Funcarama2

Funcarama3

Funcarama4: The Final Frontier

Notes About the finally Block

Funcfurter2

The SEH Termination Sample Application

Chapter 24: Exception Handlers and Software Exceptions

Understanding Exception Filters and Exception Handlers by Example

Funcmeister1

Funcmeister2

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

Pearson

Table of Contents

GetExceptionInformation

Software Exceptions

Chapter 25: Unhandled Exceptions, Vectored Exception Handling, and C++ Exceptions

Inside the UnhandledExceptionFilter Function

Action #1: Allowing Write Access to a Resource and Continuing Execution

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

UnhandledExceptionFilter 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

Chapter 26: Error Reporting and Application Recovery

The Windows Error Reporting Console

Programmatic Windows Error Reporting

Disabling Report Generation and Sending

Customizing All Problem Reports Within a Process

Creating and Customizing a Problem Report

Creating a Custom Problem Report: WerReportCreate

Setting Report Parameters: WerReportSetParameter

Adding a Minidump File to the Report: WerReportAddDump

Adding Arbitrary Files to the Report: WerReportAddFile

Modifying Dialog Box Strings: WerReportSetUIOption

Submitting a Problem Report: WerReportSubmit

Closing a Problem Report: WerReportCloseHandle

The Customized WER Sample Application

Automatic Application Restart and Recovery

Pearson

Table of Contents

Automatic Application Restart

Support for Application Recovery

Part VI: Appendixes

Appendix A: The Build Environment

The CmnHdr.h Header File

Microsoft Windows Version Build Option

Unicode Build Option

Windows Definitions and Warning Level 4

The pragma message Helper Macro

The chINRANGE Macro

The chBEGINTHREADEX Macro

DebugBreak Improvement for x86 Platforms

Creating Software Exception Codes

The chMB Macro

The chASSERT and chVERIFY Macros

The chHANDLE_DLGMSG Macro

The chSETDLGICONS Macro

Forcing the Linker to Look for a (w)WinMain Entry-Point Function

Support XP-Theming of the User Interface with pragma

Appendix B: Message Crackers, Child Control Macros, and API Macros

Message Crackers

Child Control Macros

API Macros

Index

About the Authors

Pearson