

Inside Windows® Debugging

Practical Debugging
and Tracing Strategies



Tarik Soulami

Inside Windows® Debugging: Practical Debugging and Tracing Strategies

Use Windows debuggers throughout the development cycle—and build better software

Rethink your use of Windows debugging and tracing tools—and learn how to make them a key part of test-driven software development. Led by a member of the Windows Fundamentals Team at Microsoft, you'll apply expert debugging and tracing techniques—and sharpen your C++ and C# code analysis skills—through practical examples and common scenarios. Learn why experienced developers use debuggers in every step of the development process, and not just when bugs appear.

Discover how to:

- Go behind the scenes to examine how powerful Windows debuggers work
- Catch bugs early in the development cycle with static and runtime analysis tools
- Gain practical strategies to tackle the most common code defects
- Apply expert tricks to handle user-mode and kernel-mode debugging tasks
- Implement postmortem techniques such as JIT and dump debugging
- Debug the concurrency and security aspects of your software
- Use debuggers to analyze interactions between your code and the operating system
- Analyze software behavior with the Event Tracing for Windows (ETW) framework



Get code samples on the web

Ready to download at
<http://go.microsoft.com/fwlink/?Linkid=245713>

For **system requirements**, see the Introduction.

microsoft.com/mspress

ISBN: 978-0-7356-6278-0



U.S.A. \$39.99

Canada \$41.99

[Recommended]

Programming/Windows Debugging



About the Author

Tarik Soulami, a principal development lead on the Windows Fundamentals Team, has more than 10 years of experience designing and developing system-level software at Microsoft. Before joining the Windows team, Tarik spent several years on the Common Language Runtime (CLR) Team, where he helped shape early releases of the Microsoft® .NET framework.

DEVELOPER ROADMAP

Start Here!

- Beginner-level instruction
- Easy to follow explanations and examples
- Exercises to build your first projects



Step by Step

- For experienced developers learning a new topic
- Focus on fundamental techniques and tools
- Hands-on tutorial with practice files plus eBook



Developer Reference

- Professional developers; intermediate to advanced
- Expertly covers essential topics and techniques
- Features extensive, adaptable code examples



Focused Topics

- For programmers who develop complex or advanced solutions
- Specialized topics; narrow focus; deep coverage
- Features extensive, adaptable code examples



Microsoft®

Inside Windows Debugging

Table of Contents

Cover

Title Page

Copyright Page

Contents at a Glance

Contents

Foreword

Introduction

Who Should Read This Book

Code Samples

Acknowledgments

Errata & Book Support

We Want to Hear from You

Stay in Touch

Part 1: A Bit of Background

Chapter 1: Software Development

Windows Evolution

Windows Release History

Supported CPU Architectures

Windows Build Flavors

Windows Servicing Terminology

Windows Architecture

Kernel Mode vs. User Mode

User-Mode System Processes

User-Mode Application Processes

Table of Contents

Low-Level Windows Communication Mechanisms

Windows Developer Interface

Developer Documentation Resources

WDM, KMDF, and UMDF

The NTDLL and USER32 Layers

The Win32 API Layer

The COM Layer

The CLR (.NET) Layer

Microsoft Developer Tools

The Windows DDK (WDK)

The Windows SDK

Summary

Part 2: Debugging for Fun and Profit

Chapter 2: Getting Started

Introducing the Debugging Tools

Acquiring the Windows Debuggers Package

Acquiring the Visual Studio Debugger

Comparing the WinDbg and Visual Studio Debuggers

User-Mode Debugging

Debugging Your First Program with WinDbg

Listing the Values of Local Variables and Function Parameters

Source-Level Debugging in WinDbg

Symbol Files, Servers, and Local Caches

Caching Symbols Offline for WinDbg

Troubleshooting Symbol Resolution Issues in WinDbg

Name Decoration Considerations

Getting Help for WinDbg Commands

Kernel-Mode Debugging

Your First (Live) Kernel Debugging Session

Setting Up a Kernel-Mode Debugging Environment Using Physical Machines

Setting Up a Kernel-Mode Debugging Environment Using Virtual Machines

Table of Contents

- Diagnosing Host/Target Communication Issues
- Understanding the KD Break-in Sequence
- Controlling the Target in the Kernel Debugger
- Setting Code Breakpoints in the Kernel Debugger
- Getting Help for WinDbg Kernel Debugging Commands

Summary

Chapter 3: How Windows Debuggers Work

User-Mode Debugging

- Architecture Overview
- Win32 Debugging APIs
- Debug Events and Exceptions
- The Break-in Sequence
- Setting Code Breakpoints
- Observing Code Breakpoint Insertion in WinDbg

Kernel-Mode Debugging

- Architecture Overview
- Setting Code Breakpoints
- Single-Stepping the Target
- Switching the Current Process Context

Managed-Code Debugging

- Architecture Overview
- The SOS Windows Debuggers Extension

Script Debugging

- Architecture Overview
- Debugging Scripts in Visual Studio

Remote Debugging

- Architecture Overview
- Remote Debugging in WinDbg
- Remote Debugging in Visual Studio

Summary

Chapter 4: Postmortem Debugging

Table of Contents

Just-in-Time Debugging

- Your First JIT Debugging Experiment
- How Just-in-Time Debugging Works
- Using Visual Studio as Your JIT Debugger
- Run-Time Assertions and JIT Debugging
- JIT Debugging in Session 0

Dump Debugging

- Automatic User-Mode, Crash-Dump Generation
- Analyzing Crash Dumps Using the WinDbg Debugger
- Analyzing Crash Dumps in Visual Studio
- Manual Dump-File Generation
- Time Travel Debugging
- Kernel-Mode Postmortem Debugging

Summary

Chapter 5: Beyond the Basics

Noninvasive Debugging

Data Breakpoints

- Deep Inside User-Mode and Kernel-Mode Data Breakpoints
- Clearing Kernel-Mode Data Breakpoints
- Execution Data Breakpoints vs. Code Breakpoints
- User-Mode Debugger Data Breakpoints in Action: C++ Global Objects and the C Runtime Library
- Kernel-Mode Debugger Data Breakpoints in Action: Waiting for a Process to Exit
- Advanced Example: Who Is Changing a Registry Value?

Scripting the Debugger

- Replaying Commands Using Debugger Scripts
- Debugger Pseudo-Registers
- Resolving C++ Template Names in Debugger Scripts
- Scripts in Action: Listing Windows Service Processes in the Kernel Debugger

WOW64 Debugging

- The WOW64 Environment
- Debugging of WOW64 Processes

Table of Contents

Windows Debugging Hooks (GFLAGS)

- Systemwide vs. Process-Specific NT Global Flags

- The GFLAGS Tool

- The !gflag Debugger Extension Command

- Impact of User-Mode Debuggers on the Value of the NT Global Flag

- The Image File Execution Options Hooks

Summary

Chapter 6: Code Analysis Tools

Static Code Analysis

- Catching Your First Crashing Bug Using VC++ Static Code Analysis

- SAL Annotations

- Other Static Analysis Tools

Runtime Code Analysis

- Catching Your First Bug Using the Application Verifier Tool

- A Behind-the-Scenes Look: Verifier Support in the Operating System

- The !avrf Debugger Extension Command

- The Application Verifier as a Quality Assurance Tool

Summary

Chapter 7: Expert Debugging Tricks

Essential Tricks

- Waiting for a Debugger to Attach to the Target

- Breaking on DLL Load Events

- Debugging Process Startup

- Debugging Child Processes

More Useful Tricks

- Debugging Error-Code Failures

- Breaking on First-Chance Exception Notifications

- Freezing Threads

Kernel-Mode Debugging Tricks

- Breaking on User-Mode Process Creation

- Debugging the Startup of User-Mode Processes

Table of Contents

- Breaking on DLL Load Events
- Breaking on Unhandled SEH Exceptions
- Freezing Threads

Summary

Chapter 8: Common Debugging Scenarios, Part 1

Debugging Access Violations

- Understanding Memory Access Violations
- The !analyze Debugger Extension Command

Debugging Heap Corruptions

- Debugging Native Heap Corruptions
- Debugging Managed (GC) Heap Corruptions

Debugging Stack Corruptions

- Stack-Based Buffer Overruns
- Using Data Breakpoints in Stack Corruption Investigations
- Reconstructing Call Frames from Corrupted Stacks

Debugging Stack Overflows

- Understanding Stack Overflows
- The kf Debugger Command

Debugging Handle Leaks

- A Handle Leak Example
- The !htrace Debugger Extension Command

Debugging User-Mode Memory Leaks

- Detecting Resource Leaks Using the Application Verifier Tool
- Investigating Memory Leaks Using the UMDH Tool
- Extending the Strategy: A Custom Reference Stack-Trace Database

Debugging Kernel-Mode Memory Leaks

- Kernel Memory Basics
- Investigating Kernel-Mode Leaks Using Pool Tagging

Summary

Chapter 9: Common Debugging Scenarios, Part 2

Debugging Race Conditions

Table of Contents

Shared-State Consistency Bugs

Shared-State Lifetime Management Bugs

DLL Module Lifetime-Management Bugs

Debugging Deadlocks

Lock-Ordering Deadlocks

Logical Deadlocks

Debugging Access-Check Problems

The Basic NT Security Model

Windows Vista Improvements

Wrapping Up

Summary

Chapter 10: Debugging System Internals

The Windows Console Subsystem

The Magic Behind printf

Handling of Windows UI Events

Handling of the Ctrl+C Signal

Anatomy of System Calls

The User-Mode Side of System Calls

The Transition into Kernel Mode

The Kernel-Mode Side of System Calls

Summary

Part 3: Observing and Analyzing Software Behavior

Chapter 11: Introducing Xperf

Acquiring Xperf

Your First Xperf Investigation

Devising an Investigation Strategy

Collecting an ETW Trace for the Scenario

Analyzing the Collected ETW Trace

Xperfs Strengths and Limitations

Summary

Chapter 12: Inside ETW

Table of Contents

ETW Architecture

ETW Design Principles

ETW Components

The Special NT Kernel Logger Session

Configuring ETW Sessions Using Xperf

Existing ETW Instrumentation in Windows

Instrumentation in the Windows Kernel

Instrumentation in Other Windows Components

Understanding ETW Stack-Walk Events

Enabling and Viewing Stack Traces for Kernel Provider Events

Enabling and Viewing Stack Traces for User Provider Events

Diagnosing ETW Stack-Trace Issues

Adding ETW Logging to Your Code

Anatomy of ETW Events

Logging Events Using the ETW Win32 APIs

Boot Tracing in ETW

Logging Kernel Provider Events During Boot

Logging User Provider Events During Boot

Summary

Chapter 13: Common Tracing Scenarios

Analyzing Blocked Time

The CSwitch and ReadyThread ETW Events

Wait Analysis Using Visual Studio 2010

Wait Analysis Using Xperf

Analyzing Memory Usage

Analyzing High-Level Memory Usage in a Target Process

Analyzing NT Heap Memory Usage

Analyzing GC Heap (.NET) Memory Usage

Tracing as a Debugging Aid

Tracing Error Code Failures

Tracing System Internals

Summary

Table of Contents

Appendix A: WinDbg User-Mode Debugging Quick Start

- Starting a User-Mode Debugging Session
- Fixing the Symbols Path
- Fixing the Sources Path
- Displaying the Command Line of the Target Process
- Control Flow Commands
- Listing Loaded Modules and Their Version
- Resolving Function Addresses
- Setting Code (Software) Breakpoints
- Setting Data (Hardware) Breakpoints
- Switching Between Threads
- Displaying Call Stacks
- Displaying Function Parameters
- Displaying Local Variables
- Displaying Data Members of Native Types
- Navigating Between Call Frames
- Listing Function Disassembly
- Displaying and Modifying Memory and Register Values
- Ending a User-Mode Debugging Session

Appendix B: WinDbg Kernel-Mode Debugging Quick Start

- Starting a Kernel-Mode Debugging Session
- Switching Between CPU Contexts
- Displaying Process Information
- Displaying Thread Information
- Switching Process and Thread Contexts
- Listing Loaded Modules and Their Version
- Setting Code (Software) Breakpoints Inside Kernel-Mode Code
- Setting Code (Software) Breakpoints Inside User-Mode Code
- Setting Data (Hardware) Breakpoints

Table of Contents

Ending a Kernel-Mode Debugging Session

Index

About the Author