

How Debuggers Work

Algorithm, Data Structures, and Architecture

Jonathan B. Rosenberg

Wiley Computer Publishing

JOHN WILEY & SONS, INC.

Executive Publisher: Katherine Schowalter

Editor: Marjorie Spencer

Managing Editor: Susan Curtin

Text Design & Composition: Pronto Design & Production Inc.

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Inc. is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

This text is printed on acid-free paper.

Copyright ©1996 by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional service. If legal advice or other expert assistance is required, the services of a competent professional person should be sought.

Reproduction or translation of any part of this work beyond that permitted by section 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc.

Library of Congress Cataloging-in-Publication Data:

Rosenberg, Jonathan B., 1956-

How Debuggers work: algorithms, data structure, and architecture
/ Jonathan B. Rosenberg.

p. cm.

Includes bibliographical references and index.

ISBN 0-471-14966-7 (alk. paper)

1. Debugging in computer science. 2. Computer algorithms.

3. Data structures (Computer science) 4. Computer architecture

I. Title.

QA76.9.D43R67 1996

96-9195

005.1'4—dc20

CIP

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

关于作者

Jonathan B. Rosenberg, 作者

Jonathan B. Rosenberg 是 Borland International 的 C++ 和因特网产品开发部门 VP。他负责 C++ 和与之相关的产品：包括构建集成开发环境（IDE）、编译器、链接器及相关工具，ObjectWindowsLibrary，和诸如 CodeGuard、TASM 及其工具这样的伴生产品。他的 C++ 团队也构建那些会被其它 Borland 产品（如编辑器、调试器、编译器后端、Delphi 链接器）所使用的组件。他在 Borland 最初角色是调试器的项目经理，负责 Windows 和 OS/2 平上 C++ 和 Pascal 所使用的调试器技术。那时他还负责英特网产品开发（如当时正在开发中的 Java 工具 Latte）。

在加入 Borland 之前，他在 1988 年到 1992 年是 MasPar Computer Corporation 公司的工具链经理。做为 MasPar 的最初创办人之一，他是 MasPar 编程环境（用 Smalltalk 和 C++ 构建）的架构师。在加入 MasPar 之前，他在 1986 年到 1988 年是杜克大学（Duke University）计算机科学系教授。从 1982 年到 1986 年，他是南卡罗来纳微电子中心（Microelectronics Center of North Carolina）在研究三角园区（Research Triangle Park）的设计研究和技术部门主任。MCNC（曾经）是一个 100 人左右的研究机构，致力于超大规模集成电路芯片设计。Jonathan 于 1983 年 5 月在杜克大学 Durham, NC 的计算机科学系完成了他的 Ph.D。他的数学学士学位是 1978 年在 Kalamazoo 大学获得的。Jonathan 出生于密歇根州的伯明翰。

Steven Correll, 贡献者

Steven Correll 在研究结构和商业机构中都开发过编译器、调试器和运行时系统，包括 Lawrence Livermore 国家实验室，Borland International 公司和一些创业公司（包括 MIPS Computer System 公司）。

前言

谁应该读这本书？

这本书是为那些关心调试器如何工作的人写的。它也是为那些从事创建新调试器或相似工具的勇敢者写的。甚至对于并非编写调试器的开发者，这本书也是值得一看的。每一个开发人员已经和将要在这样或那样的调试器上花费大量的时间。如果你理解这个复杂的工具——调试器到底是什么，你就可以使用它并理解它要告诉你什么，以及为什么这样更好。调试器，非常复杂和讨厌的，与 CPU 有很多的直接交互，并且和操作系统有非常紧密的关系。它必须结合编译器、链接器和其它应用程序开发工具一起来创建。对系统技术感兴趣的人可以在本书中发现大量有用信息。最后，任何对算法感兴趣的人，可以在这里发现很多调试器在实现它们功能时使用的独特而有趣的算法。

你会从本书中获得什么？

你会看到硬件是如何参与支持调试的。这很重要，因为调试变得原来越必要，并且调试器的需求和它们所能提供的功能都在持续增长。与硬件发展相近似，在过去的 10 年中，操作系统以及它们对调试器的支持和理解都有着巨大的发展。但是很明显的——并且我将说明为什么——调试当前所需的各种软件需要为调试器开发出更加优秀的功能，而为了满足调试器的这些需求，操作系统的发展还有很长的路要走。我会让你更好的理解这些重要而复杂的工具是如何工作的——我将从内部详细的说明。如果你将要创建一个调试器——从一个仅供你自己专用的简单调试工具，到高性能、产品级质量、大规模市场应用的调试器——我尝试将众多调试器技术的精华写入这个文档。最后，你可以学到一些应用在调试器中的非常有趣，并且有时候非常聪明的算法。在任何可能的地方，我都会系统的，将最有趣最重要的调试器算法展示出来。

本书布局

- 第 1 章，介绍和覆盖调试器的基本原理和它们所操作的环境。
- 第 2 章，概要介绍调试器架构——什么是用户接口和范例，调试器是如何组成的，以及各组件是如何交互的。
- 第 3 章和第 4 章，介绍基本的底层基础架构：调试器为了能够提供多数基本功能，所需要的硬件和操作系统支持。
- 第 5 章，从细节上研究调试器是如何来控制一个子进程（通常我们称为被调试体）的执行的。
- 第 6 章，接着来探究断点和单步跟踪后面的算法和数据结构——这是调试器用来控制被调试体执行的最重要的两个途径。断点就像置入运行中程序的探针，用来在操作中对其进行观察（并且不会预先修改该程序）。单步跟踪是通过一步一步观察程序行为

和控制流来了解程序运行过程的一种方法。

- 第7章和第8章，会关注在观察程序上下文的三个方面上。首先，我会检验栈调用轨迹，接下来我会研究为什么和如何反汇编硬件指令并如何将他们与用户的源代码关连起来。最后我会详细的研究一个非常大的话题，即检查被调试体中的程序变量。这包括符号表查找、范围解析、地址映射、表达式求值、函数求值，和其它更多内容。
- 第9章，会涉及一个负责且异常重要的问题——多线程调试。现代操作系统提供了“尽力而为”的线程功能，并很快在多线程中遭遇到严重的编程和调试问题。调试器还可以帮你解决由此导致的复杂的多线程错误。
- 第10章，涉及到调试GUI（图形用户界面）程序时的特殊情况。因为多数应用程序都会有一个用户界面组件，并且大多数现代GUI系统都是事件驱动的，这里面会涉及到大量重要而普遍的问题。
- 第11章，关注于对调试器（或调试器相关工具）的特殊用途，例如内存污染（memory corruption）调试，反向执行，在调试器上安装钩子代码，远程调试，在并行架构机器上调试，以及调试分布式对象。
- 第12章，涉及与调试优化代码相关的非常复杂的问题。微处理器工业界趋向于越来越多的采用RISC处理器（甚至Intel也正在向这个方向转移），并且RISC需要越来越好的编译器优化来达到它性能目标。这会给调试器带来越来越大的压力来处理这些优化，这样就可以避免仅仅为了调试一个应用程序就去掉所有的优化。

致谢

Steve Correll 编写了第 7 章和第 8 章中关于变量检查、符号表的最初版本，并就栈调用轨迹、反汇编和调试优化代码等章节做出了贡献。他对调试器和编译器之间的接口具有非常丰富的知识，没有他的帮助我无法完成本书中的那些部分。

Steve Bird 给本书起了名字并多次将我推入海中游泳，这让我保持了大脑的灵敏和专注。

Paul Gross 在我甚至没有时间写书的时候，还鼓励我完成这本书。他还允许我从 Borland 产品的屏幕抓图做为用户接口的具体例子，加入到书中。

Melissa Ailla，我的助理，使她帮我将所有的事情整理和安排的井井有条。

Nan Borreson 最先帮我联系到了出版商。

Alastair Fyfe，Borland 长期的调试器权威，是一个伟大的资源。他曾经为 UNIX `ptrace()`、UNIX `/proc`、使用 `TOOLHELP` 的 Windows 3.1，以及 Windows 95 和 Windows NT 进行过调试器相关的工作。他处理过这些系统上的各种怪异问题，并获得了大量的关于调试器的宝贵知识。他帮我复查了本书中若干章节的技术内容，极大的提升了本书的准确性。

Simon Field 从事过 10 年在多种调试器和性能检测工具上的开发工作。他是我所共事过的最聪明的人之一，也是最执着的人之一。

David Williams 是新一类狂热 Java 爱好者中的一个。并且他们决定要接替编程的世界，因此他可以做为很好的朋友。他和他的合作伙伴，Jean-Paul Buu-Sau，可能会因为编写了第一个 Native Java 调试器（译者注：即用 Java 编写的 Java 调试器）而获得荣誉。

Peter Christy，Kevin Redden，和其它在 Apple 的人愿意透露 Mac OS 的调试器 API，因此他们也应当被加入到本书中。

Borland C++团队在高质量程序员工具方面让我获益匪浅，并且他们让我真正见识了一个优秀（并且有趣和勇于探索）的团队。

我的妻子，Carole，是世界上最好的伙伴。虽然她对本书的话题没有什么兴趣，但是她对我本人以及这本书的完成是很有趣的。我的孩子，Zachary 和 Joanna，对我的工作很好奇也很支持，但是他们无法指出到底谁愿意阅读关于调试器的书。