<<软件调试>>

图书基本信息

书名：<<软件调试>>

13位ISBN编号：9787121064074

10位ISBN编号：7121064073

出版时间：2008-6

出版时间：电子工业出版社

作者：张银奎

页数：1006

<<软件调试>>

内容概要

围绕如何实现高效调试这一主题，本书深入系统地介绍了以调试器为核心的各种软件调试技术。
本书共30章，分为6篇。
第1篇介绍了软件调试的概况和简要历史。
第2篇以英特尔架构(IA)的CPU为例，介绍了计算机系统的硬件核心所提供的调试支持，包括异常、断点指令、单步执行标志、分支监视、JTAG和MCE等。
第3篇以Windows操作系统为例，介绍了计算机系统的软件核心中的调试设施，包括内核调试引擎、用户态调试子系统、异常处理、验证器、错误报告、事件追踪、故障转储、硬件错误处理等。
第4篇以Visual C/C++编译器为例，介绍了生产软件的主要工具的调试支持，重点讨论了编译期检查、运行期检查及调试符号。
第5篇讨论了软件的可调试性，探讨了如何在软件架构设计和软件开发过程中加入调试支持，使软件更容易被调试。
在前5篇内容的基础上，第6篇首先介绍了调试器的发展历史、典型功能和实现方法，然后全面介绍了WinDBG调试器，包括它的模块结构、工作模型、使用方法和主要调试功能的实现细节。
　本书是对软件调试技术在过去50年中所取得成就的全面展示，也是对作者本人在软件设计和系统开发第一线奋战10多年的经验总结。
本书理论与实践紧密结合，选取了大量具有代表性和普遍意义的技术细节进行讨论，是学习软件调试技术的宝贵资料，适合每一位希望深刻理解软件和自由驾驭软件的人阅读，特别是从事软件开发、测试、支持的技术人员和有关的研究人员。

<<软件调试>>

书籍目录

章节摘录

第1篇 绪论　第1章 软件调试基础　1955年，一个名叫Computer Usage Corporation（CUC）的公司诞生了，它是世界上第一个专门从事软件开发和服务的公司。
CUC公司的创始人是Elmer Kubie和John w Sheldon，他们都在IBM工作过。
从当时计算机硬件的迅速发展中，他们看到了软件方面所潜在的机遇。
CUC的诞生标志着一个新兴的产业正式起步了。
与其他产业相比，软件产业的发展速度是惊人的。
短短50几年后，我们已经难以统计世界上共有多少个软件公司，只知道一定是一个很庞大的数字，而且这个数量还在不断增大。
同时，软件产品的数量也达到了难以统计的程度，各种各样的软件已经渗透到人类生产和生活的各个领域，越来越多的人开始依赖软件工作和生活。
与传统的产品相比，软件产品具有根本的不同，其生产过程也有着根本的差异。
在开发软件的整个过程中，存在非常多的不确定性因素。
在一个软件真正完成之前，很难预计它的完成日期。
很多软件项目都经历了多次的延期，还有很多中途夭折。
直到今天，人们还没有找到一种有效的方法来控制软件的生产过程。
导致软件生产难以控制的根本原因是来源于软件本身的复杂性。
一个软件的规模越大，它的复杂度也越高。
简单来说，软件是程序（program）和文档（document）的集合，程序的核心内容便是按一定顺序排列的一系列指令（instruction）。
如果把每个指令看作一块积木，那么软件开发就是使用这些积木修建一个让CPU（中央处理器）在其中运行的交通系统。
这个系统中有很多条不同特征的道路（函数）。
有些道路只允许一辆车在上面行驶，一辆车驶出后另一辆才能进入，有些道路可以让无数辆车同时在上面飞奔。
这些道路都是单行道，只可以沿一个方向行驶。
在这些道路之间，除了明确的入口（entry）和出口（exit）之外，还可以通过中断和异常等机制从一条路飞越到另一条，另一条又可以飞转到第三条或直接飞回到第一条。
在这个系统中行驶的车辆也很特殊，它们速度很快，而且“无人驾驶”，完全不知道会跑到哪里，唯一的原则就是上了一条路便沿着它向前跑……　如果说软件的执行过程就好像是CPU在无数条道路（指令流）间飞奔，那么开发软件的过程就是设计和构建这个交通网络的过程。
其基本目标是要让CPU在这个网络中奔跑时可以完成需求（requirement）中所定义的功能。
对这个网络的其他要求通常还有可靠（reliable）、灵活（flexible）、健壮（robust）、易于维护（maintainable），可以简单地改造就能让其他类型的车辆（CPU）在上面行驶（portable）……　开发一个满足以上要求的软件系统不是一件简单的事，通常需要经历分析（analysis）、设计（design）、编码（code）和测试（test）等多个环节。
通过测试并发布（release）后，还需要维护（maintain）和支持（support）工作。
在以上环节中，每一步都可能遇到这样那样的技术难题。
在软件世界中，螺丝刀、万用表等传统的探测和修理工具都不再适用了，取而代之的是以调试器为核心的各种软件调试（Software Debugging）工具。
软件调试的基本手段有断点、单步执行、栈回溯等，其初衷就是跟踪和记录CPU执行软件的过程，把动态的瞬间凝固下来供检查和分析。
软件调试的基本目标是定位软件中存在的设计错误（bug）。
但除此之外，软件调试技术和工具还有很多其他用途。
比如，分析软件的工作原理，分析系统崩溃，辅助解决系统和硬件问题等。
概而言之，软件是通过指令的组合来指挥硬件，既简单，又复杂，充满神秘与挑战。

而软件调试是帮助人们探索和征服这个神秘世界的有力工具。
　　第1章　软件调试基础　　著名的计算机科学家Brian Kernighan曾经说过，软件调试要比编写代码困难一倍，如果你发挥出了最佳才智编写代码，那么你的智商便不足以调试这个代码。
　　另一方面，软件调试是软件开发和维护中非常频繁的一项任务，几乎在软件生命周期的每个阶段，都有很多这样那样的问题需要进行调试。
　　一方面是难度很高，另一方面是任务很多。
因此，在一个典型的软件团队中，花费在软件调试上的人力和时间通常是很可观的。
据不完全统计，一半以上的软件工程师把一半以上的时间用在软件调试上。
很多时候，调试一个软件问题可能就需要几天乃至几周的时间。
从这个角度来看，提高软件工程师的调试效率对于提高软件团队的工作效率有着重要意义。
　　本书旨在从多个角度和多个层次解析软件调试的原理、方法和技巧。
在分别深入介绍这些内容之前，本章将做一个概括性的介绍，使读者了解一个简单的全貌，为阅读后面的章节做准备。
　　1.1 简介　　这一节我们首先给出软件调试的解释性定义，而后介绍软件调试的基本过程。
　　1.1.1 定义　　首先，什么是软件调试?我们不妨从英文的原词software debug说起，debug是在bug一词前面加上词头de，意思是分离和去除bug。
　　Bug的本意就是昆虫，但至少早在19世纪时，人们就开始使用这个词来描述电子设备中的设计缺欠，著名发明家托马斯?阿尔瓦?爱迪生（1847—2—11—1931—10－18）就使用这个词来描述电路方面的设计错误。
　　关于Bug一词在计算机方面的应用流传着一个有趣的故事。
时间是在20世纪40年代，当时的电子计算机都还非常庞大，数量也非常少，主要用在军事方面。
1944年制造完成的Mark 1，1946年2月开始运行的ENIAC（Electronic Numerical Integrator AndComputer）和1947年完成的Mark Ⅱ是其中赫赫有名的几台。
Mark 1是由哈佛大学的HowardAiken教授设计，IBM公司制造的，Mark Ⅱ是由美国海军出资制造的。
与使用电子管制造的ENIAC不同，Markl和Mark Ⅱ主要是使用开关和继电器制造的。
另外，Mark l和Mark Ⅱ都是从纸带或磁带上读取指令并执行的，因此，它们不属于从内存读取和执行指令的存储程序计算机（stored—program computer）。
　　1947年9月9日，当人们测试Mark Ⅱ计算机时，它突然发生了故障。
经过几个小时的检查后，工作人员发现一只飞蛾被打死在面板F的第70号继电器中。
当把这个飞蛾取出后，机器便恢复了正常。
当时为Mark Ⅱ计算机工作的著名女科学家GraceHopper将这只飞蛾粘贴到当天的工作手册中（见图1.1），并在上面加了一行注释，“Firstactual case ofbug being found”，当时的时间是15：45。
随着这个故事的广为流传，越来越多的人开始使用Bug一词来指代计算机中的设计错误，并把Grace Hopper上登记的那只飞蛾看作是计算机历史上第一个被记录在文档（documented）中的Bug。
图1—1 计算机历史上第一个被记录在文档中的Bug在Bug一词广泛使用后，人们自然地开始使用debug这个词来泛指排除错误的过程。
关于谁最先创造和使用了这个词，目前还没有公认的说法，但是可以肯定的是，Grace Hopper在20世纪50年代发表的很多论文中就已频繁使用这个词了。
因此可以肯定地说，在20世纪50年代人们已经开始使用这个词来表达软件调试这一含义，而且一直延续到今天。
　　尽管从字面上看，debug的直接意思就是去除Bug，但它实际上包含了寻找和定位Bug。
因为去除Bug的前提是要找到Bug，如何找到Bug大都比发现后去除它要难得多。
而且，随着计算机系统的发展，软件调试已经变得越来越不像在继电器间“捉虫”那样轻而易举了。
因此，在台湾，人们把software debug翻译为软件侦错。
这个翻译没有按照英文原词的字面含义直译，超越了原本的单指“去除”的境界，融入了侦查的含义，是个很不错的意译。
　　在大陆，通常将software debug翻译为软件调试，泛指重现软件故障（failure）、定位故障根源，并

最终解决软件问题的过程。
这种理解与英语文献中对software debug的深层解释也是一致的。
如微软的计算机词典（Microsoft Computer Dictionary,FifthEdition）对debug一词的解释是： debug vb.To detect,locate, and correct logical or syntactical errors in a program ormalfunctions in hardware. 对软件调试的另一种更通俗的解释是指使用调试工具求解各种软件问题的过程，例如跟踪软件的执行过程，探索软件本身或与其配套的其他软件，或者硬件系统的工作原理等，这些过程有可能是为了去除软件缺欠，也可能不是。

1.1.2 基本过程 尽管取出那只飞虫非常轻松，但是找到它还是耗费了几个小时的时间。
因此，软件调试从一开始实际上就包含了定位错误和去除错误这两个基本步骤。
进一步讲，一个完整的软件调试过程是图1—2所示的循环过程，它由以下几个步骤组成。

第一，重现故障，通常是在用于调试的系统上重复导致故障的步骤，使要解决的问题出现在被调试的系统中。

第二，定位根源，即综合利用各种调试工具，使用各种调试手段寻找导致软件故障的根源（root cause）。
通常测试人员报告和描述的是软件故障所表现出的外在症状，比如界面或执行结果中所表现出的异常；或者是与软件需求（requirement）和功能规约（function specification）不符的地方，即所谓的软件缺欠（defect）。
而这些表面的缺欠总是由于一个或多个内在因素所导致的，这些内因要么是代码的行为错误，要么是不行为错误（该做而未做）。
定位根源就是要找到导致外在缺欠的内因。

第三，探索和实现解决方案，即根据寻找到的故障根源、资源情况、紧迫程度等设计和实现解决方案。

第四，验证方案，在目标环境中测试方案的有效性，又称为回归（regress）测试。
如果问题已经解决，那么就可以关闭问题。
如果没有解决，则回到第3步调整和修改解决方案。

图1—2软件调试过程 在以上各步骤中，定位根源常常是最困难也是最关键的步骤，它是软件调试过程的核心和灵魂。
如果没有找到故障根源，那么解决方案便很可能是隔靴搔痒，或者头痛医脚，有时似乎缓解了问题，但事实上没有彻底解决问题，甚至是白白浪费时间。

1.2 基本特征 上一节介绍了软件调试的定义和基本过程。
本节将进一步介绍它的基本特征，我们将分3个方面来讨论。

1.2.1 难度大 诚如Brian Kernighan先生所说的，软件调试是一项复杂度高、难度大的任务。
以下是导致这种复杂性的几个主要因素。

第一，如果把定位软件错误看作是一种特别的搜索问题，那么它通常是个很复杂的搜索问题。
首先，被搜索的目标空间是软件问题所发生的系统，从所包含的信息量来看，这个空间通常是很庞大的，因为一个典型的计算机系统中包含着几十个硬件部件、数千个软件模块，每个模块又包含着以KB或MB为单位的大量指令（代码）。
另一方面，这个搜索问题并没有明确的目标和关键字，通常只知道不是非常明确的外在症状，必须通过大量的分析，才能逐步接近真正的内在原因。

第二，为了探寻问题的根源，很多时候必须深入到被调试模块或系统的底层，研究内部的数据和代码。
与顶层不同，底层的数据大多是以原始形态存在的，理解和分析的难度比顶层要大。
举例来说，对于顶层看到的文字信息，在底层看到的可能只是这些文字的某种编码（ANSl或UNICODE等）。
对于代码而言，底层意味着低级语言或汇编语言，甚至机器码，因为当无法进行源代码级的调试时，我们不得不进行汇编一级的跟踪和分析。
对于通信有关的问题，底层意味着需要观察原始的通信数据包和检查包的各个部分。

另外，很多底层的数据和行为是没有文档的，不得不做大量的跟踪和分析才能摸索出一些线索和规律。
从API的角度来看，底层意味着不仅要理解API的原型和使用方法，有时还必须知道它内部是如何实现的，执行了哪些操作，这一点也论证了Brian Kernighan所说的调试要比编写代码困难。

<<软件调试>>

编辑推荐

ACM院士和调试技术先驱Jack B.Dennis教授做历史回顾计算机和操作系统领域资深专家David A.Solomon撰写序言。
调试高手笔耕三载集十余年经验成百万言篇，业内专家鼎力相助，汇五十年精华补软件界空白。
您将学习到： CPU的调试支持，包括异常、断点、单步执行、分支监视、JTAG、MCE等。
Windows操作系统中的调试设施，包括内核调试引擎、用户态调试予系统、验证器、Dr.Watson、WER、ETW、故障转储、WHEA等。
Visual C／C++编译器的调试支持，包括编译期检查、运行期检查，以及调试符号。
WinDBG调试器的发展历史、模块结构、工作模型、使用方法、主要调试功能的实现细节，以及遍布全书的应用实例。
内核调试、用户态调试、JIT调试、远程调试的原理、实现和用法。
异常的概念、分发方法、处理方法（SEH、VEH、CppEH），未处理异常，以及编译器编译异常处理代码的方法。
调试符号的作用、产生过程、存储格式和使用方法。
栈和堆的结构布局、工作原理和有关的软件问题，包括栈的自动增长和溢出，缓；中区溢出，溢出攻击，内存泄漏，堆崩溃等。
软件的可调试性和提高可调试性的方法。
此外，书中还诠释了很多较难理解的概念，思考了一系列耐人深思和具有普遍意义的问题。
本书是对软件调试技术在过去50年中所取得成就的全面展示，也是笔者本人在软件设计和系统开发第一线奋战10多年的经验总结。
本书适合每一位希望深刻理解软件和自由驾驭软件的人阅读，不论您是否直接参与软件开发和测试；不论您是热爱软件，还是憎恨软件；不论您是想发现软件中的瑕疵，还是想领略其中蕴含的智慧！
本书直面软件工程中的最困难任务——侦错 围绕软件世界中的最强大工具——调试器 全方位展示了软件调试技术的无比威力和无穷魅力 80个示例程序的源程序文件和项目文件 浏览符号文件的SymView工具 与内核调试引擎对话的KdTalker工具 直接浏览用户态转储文件的UdmpView工具 显示CPU执行轨迹（分支）的Cpuwhere工具 观察IDT、GDT和系统对象的SOZOOmer工具 本书是对软件调试技术在过去50年中所取得成就的全面展示，也是对作者本人在软件设计和系统开发第一线奋战10多年的经验总结。
全书共分6篇30章，选取了大量具有代表性和普遍意义的技术细节进行讨论，包括CPU的调试支持、操作系统的调试支持、编译器的调试支持、WinDBG及其实现等，是学习软件调试技术的宝贵资料。
该书可供各大专院校作为教材使用，也可供从事相关工作的人员作为参考用书使用。

版权说明