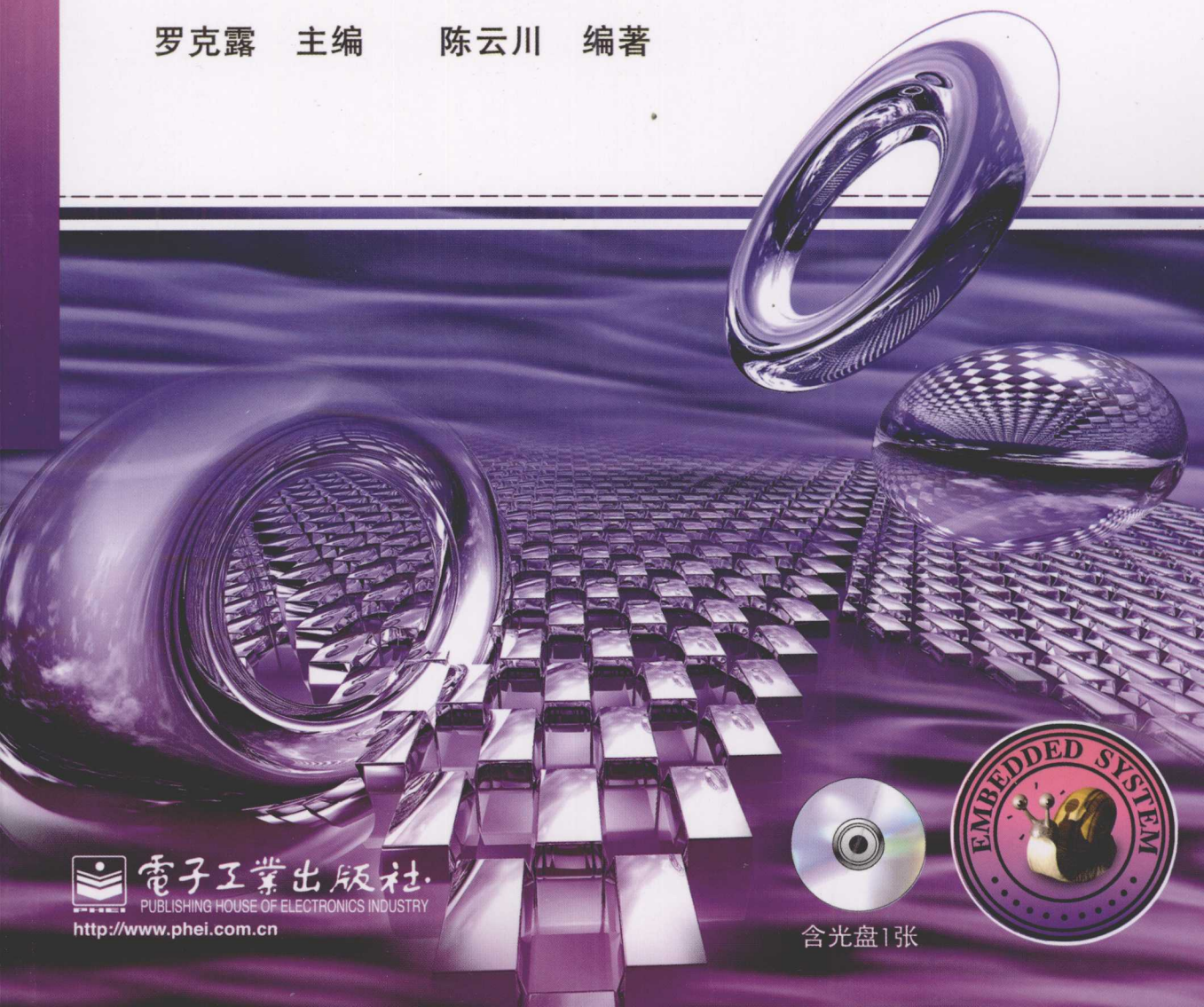
EMBEDDED SYSTEM

嵌入式技术与应用丛书

嵌入式软件调试技术

罗克露 主编　　陈云川 编著



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

含光盘1张

嵌入式技术与应用丛书

嵌入式软件调试技术

罗克露　主编
陈云川　编著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书兼顾理论与实践。全书首先对调试技术及嵌入式调试手段进行一个概览，然后重点对 Linux 环境下的嵌入式调试技术进行详细的讲解，最后给出两个综合应用实例：MPEG-4 视频播放器的设计和基于 GPS 的移动定位终端。在编写思路上，本书以理论为先、实践为重；具体到调试技巧和手段上，则全部结合具体实例展开。书中各个实例之间采取从前至后逐步深入的方式，既衔接有序，又便于读者学习。

本书配套光盘包含书中相关的补充资料、与 Sitsang 评估板相关的工具链和源代码、本书第 3～9 章的源代码，以及书中使用到的工具和软件库等。

本书既可作为 Linux 环境下嵌入式软件调试技术从入门到精通的学习用书，也可供从事 Linux 环境下的嵌入式软件调试的工程人员参考使用。

图书在版编目（CIP）数据

嵌入式软件调试技术/罗克露主编. —北京：电子工业出版社，2009.1
（嵌入式技术与应用丛书）
ISBN 978-7-121-07726-5

Ⅰ. 嵌… Ⅱ. 罗 Ⅲ. 软件—调试 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字（2008）第 175281 号

策划编辑：高买花
责任编辑：徐 磊
印 刷：北京东光印刷厂
装 订：三河市皇庄路通装订厂
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1 092 1/16 印张：30.75 字数：738 千字
印 次：2009 年 1 月第 1 次印刷
印 数：4 000 册 定价：59.00 元（含光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

出 版 说 明

嵌入式技术是 21 世纪最具生命力的新技术之一，经过近几年的快速发展，它已经成为电子信息产业中最具增长力的一个分支。随着手机、掌上电脑、GPS、机顶盒等新兴产品的大量应用，嵌入式系统的设计正成为软硬件工程师越来越关心的话题。面对不断涌现的技术需求和发展机遇，各大嵌入式系统开发商、各科研院所的研发人员都急需一套全方位、针对性强且具有实际指导意义的嵌入式技术类书籍；各高等院校相关专业的本科生、研究生也迫切希望了解、掌握嵌入式系统的开发技巧，以推动嵌入式技术在各领域的广泛应用和快速发展。

《嵌入式技术与应用丛书》正是针对当前技术与市场需求，由国内站在 IT 业前沿并有实践开发经验的嵌入式系统专家，以实用技术为主线，理论联系实际，将他们在理论研究与实践工作中积累的大量经验和体会有机地融于一体，以丛书的形式奉献给广大读者！

本丛书由基础理论类、硬件设计类、软件开发类、综合应用类书籍组成，立足当前嵌入式技术的发展趋势、核心技术及其主要应用领域，将技术热点与实践应用紧密结合，以实际应用为主线，融合关键性嵌入式设计技术，围绕嵌入式设计理论、开发流程、嵌入式软件验证及测试、代码可重构及代码优化等方面进行深入浅出的讲解和论述。

读者群定位于高等院校相关领域的高年级学生，科研、开发人员和嵌入式相关领域设计人员等，本丛书可作为嵌入式领域学习、开发人员的参考资料，也可作为高等院校相关专业师生的教学参考书。

本丛书的出版得到了业界许多专家、学者的鼎力相助，对此表示衷心的感谢！同时，热切欢迎广大读者提出宝贵意见，或者推荐更多优秀选题（gmholife@hotmail.com），共同为嵌入式技术的发展添砖加瓦！

电子工业出版社通信分社

2008 年 10 月

前　言

在程序开发过程中，一个公认的事实是编写代码并不难，但是如何写出正确的代码、如何排除代码中的错误，却并不是一件简单的事情。如何排除代码中的错误，这就涉及软件的调试。调试技术的理论基础并不复杂，但是调试本身却是一门实践性非常强的技术。

需要说明的是，嵌入式系统中的软件调试与桌面软件的调试有很大的不同。调试嵌入式系统时，调试器和被调试程序往往是物理上分离的。调试器运行在宿主机上，而被调试程序运行在目标机上，宿主机与目标机之间通过某种媒介进行通信。同时，还要在目标机上运行一个称为调试代理（Debug Agent）的监控程序，由它来负责与运行在宿主机上的调试器进行通信，控制被调试程序的执行，并将被调试程序的执行情况及时反馈给调试器。这进一步增加了调试的复杂性。

当前图书市场上有关调试技术的书籍并不多。但是，对于从事实际开发的工程人员而言，尤其是对广大的嵌入式系统程序员而言，调试是一个无法回避的永恒话题。因此本书以嵌入式开发中的调试技术和调试手段作为全书的核心，在当前众多的计算机编程开发书籍中另辟蹊径，独树一帜，对于刚接触嵌入式程序开发的读者而言，必将大有裨益。同时，对于经验丰富的程序员而言，本书也具有较大的参考价值。

本书所用平台

本书采用的嵌入式硬件平台为 Intel 公司的 Sitsang 评估板，其上运行的操作系统为 ARM-Linux 2.4.19。本书宿主机端软件平台采用 Red Hat 9.0（Linux 2.4.20-8），本地调试器为 GDB 5.3（Red Hat 9.0 默认配置），交叉调试器在 GDB 6.4 的基础上构建。另外，在本书写作的后期，作者换到了一台笔记本电脑上工作，为了与时俱进，其上运行的是 Fedora Core 操作系统。本质上讲，Fedora Core 源自 Red Hat，两者并无根本不同。为消除读者对此存疑，特此予以澄清。

尽管本书采用的软硬件平台是固定的，但是书中演示的大部分实例都不依赖于具体的软硬件环境。因此如果读者所采用的软硬件平台与作者书中采用的有所不同，也无须多虑，大部分功能和操作都是相同的。

本书内容组织

本书共分为 3 篇。

上篇基础篇，包括第 1～2 章。主要对调试的基础知识进行了大致的概览，对各种调试手段进行了简单的说明和对比。

中篇系统篇，包括第 3～7 章。第 3 章介绍 GDB 调试器的使用；第 4 章主要介绍如何通过 GDB 进行远程调试，同时也涉及了远程调试可能要用到的一些工具，如 NFS、minicom 等；第 5～7 章主要介绍在一些特定开发场景下所采用的调试技术，包括网络环境下的调试、多线程与多进程环境下的调试、静态库与动态库的调试，等等。

下篇应用篇，包括第 8～9 章。这部分内容属于较综合的话题，运用到了前述章节中

所讲述的一些内容。第 8 章围绕 MPEG-4 视频流的解码和播放展开，而第 9 章则给出了一个基于 GPS 的移动定位终端的开发。

本书配套光盘内容及使用方法

本书配套光盘包含书中相关的补充资料、与 Sitsang 评估板相关的工具链和源代码、本书第 3～9 章的源代码，以及书中使用到的工具和软件库等。

使用时只需将本光盘放入光盘驱动器中，选择各文件夹浏览即可。

本书读者对象

- Linux 编程初学者
- 通过 GDB 调试程序的开发人员
- 基于 Linux 的嵌入式系统开发人员

致谢

本书由罗克露教授主编，陈云川编著，参编人员还有王治国、冯强、曾德惠、许庆华、程亮、周聪、黄志平、胡松、邢永锋、邵军、边海龙、刘达因、赵婷、马鸿娟、侯桐、赵光明、李胜、李辉、侯杰、王红研、王磊、闫守红、康涌泉、李欢、蒋杼倩、王小东、张森、张正亮、宋利梅、何群芬、程瑶等，在此表示衷心的感谢。

在本书的写作过程中，得到了电子科技大学电子工程学院伍瑞卿老师和顾庆水老师的无私指导和帮助，得以使用 Intel PCA 联合实验室的 Sitsang 评估板作为本书所使用的硬件平台。在此向两位老师表示衷心的感谢。

本书的写作也得到了电子科技大学计算机学院实时系统教研室廖勇博士的大力支持和帮助，在此致以诚挚的谢意。

还要感谢的朋友包括陈兆荣、刘国霞、黄伟、刘峰、马俊、赵微、丁熠、骆张强，以及电子科技大学实时系统教研室多媒体组的所有成员和网友 twentyone。

特别要感谢亲爱的陈静女士在本书写作过程中所给予的理解和支持，陪我一起走过了很多艰苦之旅。同时，在“5·12”汶川大地震发生之后，对于编者前往绵竹市从事志愿者工作的行为给予了极大支持，一个人度过了很多个担惊受怕的夜晚。谢谢您，静！

由于时间仓促，书中疏漏之处在所难免，欢迎广大读者和同行批评指正。

配套服务

为充分展现本书编写特点，帮助读者深刻理解本书编写意图与内涵，进一步提高对本书教学的使用效率，我们建立本书使用指导联络方式，是读者与编者之间交流沟通的直通车。欢迎读者将图书使用过程中的问题与各种探讨、建议反馈给我们，本书编者会竭诚给您满意的答复。我们的联系方式是 E-mail：china_54@tom.com。

目 录

上篇 基础篇

中篇 系统篇

上篇　基础篇

软件调试概述

边界扫描测试技术（JTAG）

第1章　软件调试概述

知识点

- 什么是软件调试
- 什么是调试器
- 软件调试的分类
- 软件断点与硬件断点
- 调试器应当遵循的原则
- 嵌入式软件调试手段

本章导读

本章首先讨论什么是软件调试，什么是调试器。继而对比了桌面环境下的调试模型与嵌入式环境下的调试模型之间的异同。紧接着说明了软件调试的分类。断点对于调试至关重要，因此专门用了一小节说明两种类型的断点——软件断点和硬件断点。在简略地列举了调试器应当遵循的几项原则之后，对几种嵌入式软件调试手段做了说明，重点是 JTAG 调试的原理。

1.1　什么是软件调试

尽管调试技术在通常情况下是对计算机领域而言的，但它并不是计算机领域的专利，调试也常常发生在 CPU 与存储器之外的场合。著名的计算机科学家塔嫩鲍姆（Andrew S. Tanenbaum）教授曾在他的经典著作[1]里把程序比喻为菜谱和原料的有机组合。他这样写到，“想象一位有一手好厨艺的计算机科学家正在为他的女儿烘制生日蛋糕。他有做生日蛋糕的食谱，厨房里有所需的原料：面粉、鸡蛋、糖和香草汁等。做蛋糕的食谱即是程序（即用适当形式描述的算法），计算机科学家就是 CPU，而做蛋糕的各种原料就是输入的数据”。这里为了说明调试的原理，现在假设这位计算机科学家并没有一手好厨艺（事实上这更接近客观情况一些，尽管塔嫩鲍姆教授可能是一个特例），同时菜谱本身因为年代久远（很可能是从这位计算机科学家的祖母那里传下来的），其中的某些步骤已经不正确了。那么理所当然地，当这位计算机科学家做出生日蛋糕的第一版时，可能会发现蛋糕并没有预期的那么好，蛋糕显得太甜了，而计算机科学家的女儿并不喜欢过于甜腻的食品（很可能是因为电视上正在宣传过量食用甜食的害处）。无论如何，计算机科学家需要重新做一个更好的生日蛋糕。这一次，每做完食谱上的一个步骤之后，计算机科学家都会用汤匙尝一下处于半成品状态的蛋糕是否过甜了。由于计算机科学家是一位很细心的人，因此他很快就发现问

题出在香草汁上。半个世纪前，当计算机科学家的祖母为孩子们做生日蛋糕时，那个时候的香草汁还没有加入蔗糖。可是，计算机科学家现在厨房里放的香草汁却是加了 30%蔗糖的。于是，计算机科学家决定减少糖的用量，因为香草汁里面已经含有足够多的糖了。这样，通过一步一步的“跟踪调试”，一个漂亮、美味的生日蛋糕就制作完成了。

在上面这个假想的场景中，计算机科学家在完成生日蛋糕第二版的过程中，在食谱中的每一个步骤上停下来，用汤匙品尝蛋糕是否过甜，最后发现问题所在的整个过程，就是调试。计算机科学家在这里采用的“调试”手段有两种：“单步执行”和“观察变量”。除了厨房里的调试之外，这里再假想一个需要“调试”的场景。假设一位驯兽师正在训练一只海豚表演水上钻火圈，尽管这只海豚很聪明，可是驯兽师还是发现当它跃过火圈的时候，动作有点不够流畅。由于海豚跃过火圈的动作很快，并且驯兽师也没有办法让海豚在刚好穿过火圈的时候停下来，所以驯兽师无法看清越过火圈时的细节。于是驯兽师采用了这样一个办法，他在火圈侧面安装了一个数字摄影机。当海豚跃过火圈的时候，摄影机将把全过程录制下来。这样，驯兽师就可以在一台数字播放机或个人计算机上观察海豚完成特技动作的整个过程，可以随时定格画面、回退或者快进，以方便地观察海豚所做的特技动作中的每一个细节。最后，驯兽师发现问题出在海豚的尾鳍上。在这个假想的场景中，驯兽师所做的工作也可以看成是一种“调试”。

不过，厨房里或海洋馆里的“调试”并不是本书关心的重点，本书关心的是计算机系统中的调试技术。一般地，可以给软件调试进行如下的定义。

1. 定义 1.1

软件调试是为了发现并排除软件程序中的错误，而通过某种方法控制被调试程序的执行过程，以便随时查看和修改被调试程序执行状态的方法。

在一个具体的程序中，指令与数据从来就是一个不可分割的整体。控制程序的执行过程和查看程序执行状态是任何调试工具和调试手段都必须提供的两个功能。通常情况下，控制程序执行状态的方法有单步执行、设置断点、从指定的地址开始执行和运行到函数（过程）返回等。所有这些方法中，最重要的手段是设置断点。如果说控制程序执行过程是针对指令的调试手段的话，那么查看被调试程序的执行状态实际上就是针对数据所提供的调试手段。这两者是密不可分的。针对数据的调试手段有设置观察点（watchpoint）、追踪点（tracepoint）、查看堆栈、查看 CPU 寄存器值和查看内存空间等。

上面给出了软件调试的简单定义，那么软件调试的目的又是什么呢？为什么需要软件调试技术呢？软件调试能够解决什么样的问题呢？其实答案很简单，程序员并非超人，除了最简单的 hello world 程序之外，任何程序员都可能犯错误（有时甚至 hello world 程序也会存在错误）。正所谓“当局者迷，旁观者清”，为了能够发现程序中的问题，程序员需要一种使自己“置身事外”的方法来观察程序的运行情况。调试技术和调试手段恰好为程序员提供了这样一种能力，使得程序员能够站在被调试程序之外来观察程序中所发生的事情，从而发现程序中存在的逻辑错误或由粗心导致的错误，防止自己迷失在程序本身的迷宫之中。但是，与物理学家们所面临的问题一样，这种“置身事外，冷眼旁观”的手段也是有限度的[2]。在 20 世纪 20 年代，量子物理学家发现了所谓的“测不准原理”，即在量子尺度上（也就是在亚原子尺寸上），观察者已经不能把自己与他们正在测量的东西割裂开了。换

句话说，观察活动本身就会对实验的结果产生影响。观察者实际已经成为了实验的一个部分。这是亚原子尺度上一个不可避免的事实。当计算机科学家们所要调试的对象在时序上的要求越来越精细、越来越严格时，他们也将会面临与被调试的对象无法割裂开来的尴尬。而这并不是由于调试工具的缺陷导致的，而是由于调试技术本身就无法避免这一问题的困扰。尽管如此，在绝大多数情况下，调试工具仍然是计算机工程人员在万不得已的最后关头使出的终极“杀手锏”。这样的调试工具通常被称为调试器（debugger）。

2. 定义 1.2

调试器（debugger）是一种软件工具，通过它可以控制程序的执行流程，并允许开发者查看和/或修改机器状态，从而发现和排除程序中存在的错误。

可以发现，定义 1.1 和定义 1.2 有些相似，但不同的是，定义 1.1 定义的是一种方法，而定义 1.2 定义的则是一种软件开发工具。

调试器提供了一个受控的执行环境，在这个环境中，程序运行过程中的一切对于程序员都是清清楚楚、明明白白的，不存在“雾里看花”的情况。被调程序成了一个受调试器控制的“傀儡”，调试器可以支配它的一切行动，随时冻结它的状态。任何时候，只要调试器大喝一声“站住”，被调程序就得乖乖地停住；而当调试器发出“继续前进”的命令之后，被调程序才能继续运行。如此这般走走停停，直到调试器认为被调程序已经可以正确工作不用再“发号施令”了为止。

通常，在桌面环境下，调试器都是像上面描述的这样工作的，也就是调试器直接控制被调程序的运行。桌面环境下的调试模型如图 1-1 所示。表面上看来，调试器直接控制被调程序的运行，但实际上，这是通过操作系统内核来完成的。比如，在 Linux 内核上，这就是通过 ptrace()系统调用来实现的[3]。

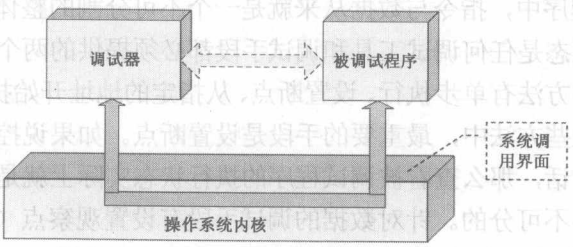


图 1-1　桌面环境下的调试模型

可是，在嵌入式环境下，一切都变了。因为嵌入式系统往往是资源受限的计算机系统，因此不仅 CPU 速度可能会比较慢，存储器空间可能也不大，甚至没有键盘鼠标等人机交互设备。这些限制导致了什么样的问题呢？由于 CPU 速度比较慢和内存空间比较小，因此可能根本无法运行调试器；其次，由于没有人机界面，即使能运行调试器，也可能无法对调试器进行操作和控制，更不要说通过调试器去控制被调试的程序了。上面给出的是硬件资源方面的一些考虑。对于软件环境而言，也存在一些限制。有相当一部分嵌入式操作系统没有提供完善的操作系统功能，可能缺少操作系统的某一项功能（如文件系统或内存管理）。典型的，如μC/OS 就只提供了基本的进程管理和内存管理，文件系统并不是默认包含的。而要想运行调试器，文件系统几乎是必不可少的。因此，直接在嵌入式系统上运行调试器

的做法非常罕见，通常的做法是采用宿主机（host）加目标机（target）的方式。在目标机上运行被调程序，宿主机则运行调试器。目标机和宿主机之间通过某种媒介通信，典型的媒介有串口、并口、以太网或 JTAG。目标机的类型多种多样，但宿主机的角色通常由 PC 担任。同时，在宿主机上还要配备一套相应的开发工具，如交叉编译器、交叉调试器、性能分析器和映像下载程序等。这一整套工具被称为工具链（Toolchain）。

3. 定义 1.3

交叉编译器：一个运行在宿主机平台上的编译程序，它编译出的程序可以在目标机上运行。

至于交叉调试器，后面再进行说明。这里之所以把编译器和调试器都冠以“交叉”（cross）二字，主要是因为在这样的开发方式下，目标机和宿主机往往属于异构的平台，因此用“交叉”二字以示与本地（local）编译器和调试器相区别。

由于目标机和宿主机在物理上是分离的，因此其调试方式也就和通常的桌面环境下的调试有了区别。目前采用得最多的是调试代理（Debug Agent）加调试器的方式。调试代理运行在目标机上，而调试器则运行在宿主机上，它们之间采用某种协议通信，比如 GDB 调试器采用 RSP（Remote Serial Protocol，远程串行协议），而 ARM 公司的调试工具 Angel 采用的则是 ADP（Angel Debug Protocol，Angel 调试规程）。采用这样的调试方式以后，真正控制程序的是调试代理，而调试代理又受宿主机上的调试器控制。

4. 定义 1.4

调试代理是在目标程序的可执行映像中加入的一个用于调试用途的特殊软件成分，它可以和运行在宿主机端的调试器通信，并根据调试器发出的指令控制目标程序的运行，同时将目标程序的执行状态反馈给调试器。

这就好比两军对阵的时候，我方在敌人的军营里面安插了一名卧底，每当我方希望控制敌人的行踪的时候，就通过某个秘密的通信渠道给卧底布置任务，由他去完成。在调试嵌入式软件的时候，调试代理扮演的就是“卧底”的角色，它不仅要和自己的上司——调试器联络、接受任务，而且还要控制被调程序的执行过程，并把结果反馈给调试器。嵌入式环境下的调试模型如图 1-2 所示。

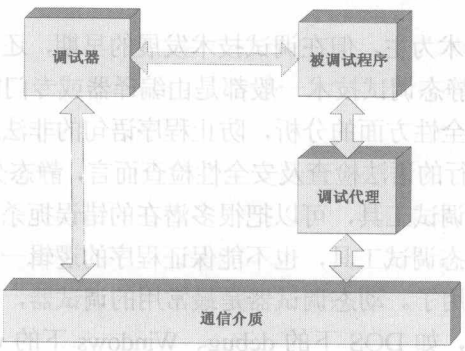


图 1-2　嵌入式环境下的调试模型

对比图 1-1 和图 1-2 可以发现，两种调试模型在结构上其实有很相似的地方，那就是抽象地看，调试器对被调程序的控制，都是以某种通信方式来实现的。然而两者间的区别

也很明显，在桌面环境下，这种通信过程由位于底层的操作系统内核完成；而嵌入式环境下的调试则是经由某种通信介质来完成的，并且还在被调程序与通信介质之间插入了一层调试代理，调试器与调试代理的通信需要遵守某种通信协议。

1.2 软件调试的分类

按照不同的标准，可以对软件调试方法进行不同的分类，如表 1-1 所示。如果以调试所处的阶段来分，可以分为静态调试和动态调试两类；如果以调试器所处理的指令粒度来分，可以分为机器级调试和源码级调试；如果以被调程序是运行在用户空间还是内核空间来分，可以分为任务级调试和系统级调试；如果以被调程序是否与调试器位于同一运行环境来分，可以分为本地调试和远程（交叉）调试等，具体的分类方法多种多样。

表 1-1 软件调试的一个简单分类

分类方法	类别 1	类别 2
调试所处阶段	静态调试	动态调试
处理的指令粒度	机器级调试	源码级调试
调试对象所在空间	任务级调试	系统级调试
运行环境是否异构	本地调试	远程（交叉）调试

值得说明的是，这里的分类主要是针对具体的调试手段而言的。对于调试器而言，这样的分类界限却很模糊。比如，以本书中将要重点讲述的 GDB 调试器为例，尽管多数情况下是作为源码级调试器被使用，但是它也同样可以用于机器级调试；同时，尽管通常情况下它是一个任务级调试器，但是对于打了 KGDB 补丁的内核而言，它又是一个系统级调试器；至于本地调试与远程调试的区分，则更是显得模糊，GDB 独特的设计理念使得它在成为 UNIX 平台下标准的本地调试工具的同时，也成了应用频繁的远程调试器。下面将对这些分类逐一介绍。

1.2.1 静态调试和动态调试

本书以运行时调试技术为主，但在调试技术发展的早期，还有一种对程序源代码进行静态分析的调试技术[4]。静态调试技术一般都是由编译器或专门的静态分析工具完成的，它对源代码进行语法及安全性方面的分析，防止程序语句的非法跳转或引用未初始化的变量等。相对于编译器所进行的语法检查及安全性检查而言，静态分析/调试工具所做的检查更为严厉。通过采用静态调试工具，可以把很多潜在的错误扼杀在摇篮之中。

但是，即使使用了静态调试工具，也不能保证程序的逻辑一定是对的。这时，就该轮到动态调试器发挥它的作用了。动态调试器是最常用的调试器，通常看到的调试器基本上都属于动态调试器的范畴，如 DOS 下的 debug、Windows 下的 windbg、Linux 下的 GDB 等。动态调试器也常被称为“断点”调试器，它与用户的交互方式有两种：命令行接口或图形用户接口。DOS 下的 debug 调试器就只提供命令行接口，而 Linux 下的 GDB 调试器则既有命令行接口，也有图形用户接口（ddd 就是常用的 GDB 图形前端）[5]。用户通过发出各种调试命令来控制被调程序的执行。一般地，动态调试器都采用在被调程序中插入断

点的方式实现对被调程序的控制，当被调程序运行到断点位置时，会触发一个中断或异常，在中断处理或异常处理程序中用户就可以检测被调程序的各种状态，如变量值、寄存器值、函数堆栈和函数调用层次等。动态调试器最常见的用途是用于循环调试（cyclical debugging）。所谓循环调试，是指在程序发生错误之后，在可疑的地方设置断点，反复运行被调程序，检查程序运行过程中的状态，逐步缩小可疑范围，直到找出错误的根源。

1.2.2　机器级调试与源码级调试

除此之外，还可以从调试器所“看见”的指令粒度（granularity of instruction）上把调试器分为两类：机器级调试器和源码级调试器。

机器级调试器处理的是最底层的机器指令。通过它可以观察到计算机最基本的工作情况，可以观察到每一条指令执行后对寄存器、存储器产生的影响。机器级调试不是一件快乐的事情，如果要调试的是嵌入式系统的板级支持包（Board-level Support Package，BSP）或操作系统引导程序的话，可能就会面对机器级调试的痛苦经历。

在高级程序设计语言出现以前，程序员们能使用的编程语言只有汇编语言，自然，采用的调试手段也只有机器级调试。可是，随着高级程序设计语言的出现，程序员们开始了新的时代。高级程序设计语言提供了比汇编语言更好的抽象，底层的机器指令变成了更容易理解和使用的数学语言或公式语言（如 Pascal、C、Fortran），程序员们摆脱了直接面对所有底层硬件细节的噩梦。随后，源码级调试器也紧跟着“出世”了。源码级调试器面向高级程序设计语言，屏蔽了大量机器底层细节，允许程序员把精力集中到高级程序设计语言的框架里对程序的执行路径进行追踪。

为了实现源码级调试，调试器必须知道高级语言中的每一条语句与汇编代码的对应关系。多数现代编译器都提供了一个编译选项，用以在编译时生成调试信息。这些调试信息包含了源代码行和编译生成的汇编指令之间的对应关系。当在源代码的某一行上设置断点的时候，实际上是在对应的汇编指令上设置了一条断点指令或非法指令，同时把原来位于该位置处的指令保存到了某个地方。在触发断点之后，调试器把控制权交给用户，此时用户可以查看机器的各种状态。在用户把控制权还给调试器之后，调试器将此前保存的指令恢复，继续往下执行。

1.2.3　任务级调试与系统级调试

从调试范围来分，可以将调试分为任务级调试和系统级调试。

任务级调试运行在操作系统的支持下。任务级调试的范围仅限于任务（某些情况下等价于进程），不包括操作系统内核。

系统级调试的对象通常运行在内核空间，如驱动程序或中断处理程序（Interrupt Service Routine，ISR）。内核通常直接和底层硬件打交道，同时内核关注的重点往往和同步、互斥、共享之类精细的操作有关，一旦内核出现错误，那么整个系统就都无法继续运行了，因为内核被带到了一个不确定的状态。更糟的是，内核错误可能非常难以重现。比如，如果内核中存在一个竞争条件，那么很可能这个错误出现的概率只有万分之一，而在其余的情况下都是正确的。系统级调试往往比任务级调试要困难得多。