CN43-1258/TP
ISSN 1007-130X
计算机工程与科学
COMPUTER ENGINEERING & SCIENCE
2004 年第 26 卷第 10 期
Vol.26,No.10,2004

文章编号:1007-130X(2004)10-0100-04

GDB 远程调试及其在嵌入式 Linux 系统中的应用*

GDB Remote Debugging and Its Application in Embedded Linux Systems

郭胜超,吕　强,杨季文,钱培德
GUO Sheng-chao, Lü Qiang, YANG Ji-wen, QIAN Pei-de
(苏州大学计算机工程系,江苏 苏州 215006)
(Department of Computer Engineering, Soochow University, Soochow 215006, China)

摘　要:嵌入式 Linux 系统的研究和应用越来越热。针对如何完成系统调试工作的问题,本文首先介绍了 GDB 远程调试技术在该领域的应用概况,然后从剖析 GDB 远程调试的工作机制入手,具体描述了实现该调试手段的一般方法,重点介绍了使用 GDB 远程调试功能在嵌入式 Linux 系统中调试各类程序代码的应用实例。

Abstract: The research and application of embedded Linux systems become more and more popular. This thesis first gives the overview of the GDB remote debugging technology in embedded Linux development. After introducing the working mechanism of GDB remote debugging, the authors concretely describe the common method of applying remote debugging to embedded Linux, and demonstrate the cases of applying remote debugging to all kinds of codes.

关键词:嵌入式 Linux;远程调试;GNU 调试器

Key words: embedded Linux; remote debugging; GDB

中图分类号:TP368.2　　**文献标识码**:A

1　引言

随着人们对嵌入式系统应用需求的不断加强,嵌入式系统的体系结构和系统功能也越来越强大和复杂。微处理器制造工艺的迅猛发展,为嵌入式系统的构造提供了高性能、低功耗的中央处理器和稳定可靠的硬件架构。类似于台式机桌面应用的开发,嵌入式系统同样需要在软件方面与硬件同步发展。尽管伴随着嵌入式系统的发展也出现了一些嵌入式操作系统,如 Vxwork、Neculeus、Palm OS 和 Windows CE 等[1,2],但这些商业化的操作系统,其高昂的价格和定制能力的缺乏,使得许多低端应用无法使用它们作为系统的软件平台。另一方面,随着自由软件运动的兴起,Linux 继在桌面系统取得巨大成功之后,又以其开放源码、容易定制和扩展、多硬件平台支持和内置网络功能等优良秉性,逐渐成为嵌入式系统的研究热点和广泛使用的系统平台。同时,GNU[3]免费提供的一整套工具链,为嵌入式 Linux 系统的开发和调试提供了完整的支持,其调试器 GDB (GNU Debugger)的远程调试能力,避免了价格昂贵的仿真器工具在调试中的使用,开发人员可以在宿主机上使用 GDB 方便地对运行于目标平台

* 收稿日期:2003-04-18;修订日期:2003-09-04
作者简介:郭胜超(1978-),男,江苏镇江人,硕士生,研究方向为计算机中文信息处理及应用技术;吕强,教授,研究方向为计算机操作系统、分布式计算和计算语言学;杨季文,教授,研究方向为计算机中文信息处理;钱培德,教授,研究方向为计算机中文信息处理。
通讯地址:215006 江苏省苏州市苏州大学 158 号信箱;Tel:(0512)65226960;E-mail:scguo@zhhz.suda.edu.cn
Address: Mail Box 158, Soochow University, Soochow, Jiangsu 215006, P.R.China

上的程序进行跟踪调试。

2 GDB 的远程调试功能

2.1 嵌入式 Linux 系统调试方法概述

嵌入式系统的调试是一个很特殊的领域，不同于桌面系统程序的调试，调试器和被调试代码不可能同时运行在资源有限的目标平台上，开发人员要通过一些特殊的硬件或软件调试手段，在宿主机上使用调试器对目标代码进行监控和调试。硬件仿真器功能足够强大但价格不菲；CPU 内部集成的调试模块，如 ARM 的 JTAG(Joint Test Action Group，简称 JTAG)接口和 Motorola 系列的 BDM(Background Debug Module，简称 BDM)接口[1]，能够实时监控 CPU 的所有动作，但配套的商业软件同样昂贵，故基于硬件的调试方案并没有在嵌入式 Linux 系统开发中得到广泛应用。虽然软件调试方法不如硬件调试工具功能强大，但已经能够很好地胜任嵌入式 Linux 系统中的绝大多数调试工作，尤其适合不需要做实验板整合和硬件驱动设计的嵌入式开发。

GDB 是 GNU 免费提供的调试除错工具，可以用于 C、C++、Pascal 和 Fortran 等程序的跟踪调试。更吸引人的是，在嵌入式系统开发中，开发人员能够使用 GDB 方便地以远程调试的方式单步执行目标平台上的程序代码、设置断点、查看内存，并同目标平台交换信息。GDB 同目标机交换信息的能力相当强，胜过绝大多数的商业调试工具，甚至可以与某些低端仿真器媲美。同样，与打印输出等传统的软件调试手段相比，GDB 远程调试的动态、实时、方便等方面的优势非常明显。因而，GDB 的远程调试方法逐渐成为嵌入式 Linux 开发的首选调试方案。

2.2 GDB 远程调试的工作机制

使用 GDB 进行远程调试时，运行在宿主机上的 GDB 通过串口或 TCP 连接，与运行在目标机上的调试插桩(stub)[1,4]以 GDB 标准远程串行协议协同工作，从而实现对目标机上系统内核和上层应用的监控和调试功能。调试 stub 运行在目标系统中，作为宿主机 GDB 和目标机调试程序间的一个媒介存在。GDB 远程调试结构如图 1 所示。

为了监控和调试程序，主机 GDB 通过串行协议，使用内存读写命令，无损害地将目标程序原指令用一个 Trap 指令代替，完成断点设置动作。当

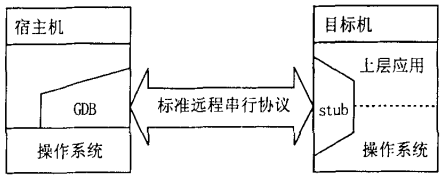

图 1 GDB 远程调试结构

目标系统执行该 Trap 指令时，stub 可以顺利获得控制权。此时，主机 GDB 就可以通过 stub 来跟踪和调试目标机程序了。调试 stub 会将当前场景传送给主机 GDB，然后接收其控制命令；stub 按照命令在目标系统上进行相应动作，从而实现单步执行、读写内存、查看寄存器内容、显示变量值等调试功能。调试 stub 由 set_debug_traps()、handle_exception()等一系列功能函数组合而成。这些函数各司其职，通过 GDB 远程串行协议与主机 GDB 通信协作，实现远程调试。GDB 远程串行协议是一种基于消息的 ASCII 码协议，它定义读写数据的信息，控制被调试程序，并报告运行程序的状态。协议交换数据的格式如下[2]：

$ < data > # [chksum]

其中，< data > 是 ASCII 码十六进制字符串，[chksum]是一个字节的十六进制校验和。调试 stub 响应主机 GDB 数据的方式有两种：若消息传送正确，则回送响应数据或 OK；否则，返回由目标平台自己定义的错误码，并由 GDB 控制台向用户报告。通信数据中的 < data > 包括了非常丰富的远程调试操作指令和状态信息，完整的定义信息可参考 GDB 参考文档 info-14 中 GDB Remote Serial Protocol 部分[3]。

3 远程调试的实现方法

远程调试环境由宿主机 GDB 和目标机调试 stub 共同构成，两者通过串口或 TCP 连接，使用 GDB 标准远程串行协议协同实现远程调试功能。若双方环境建立无误，则 stub 会首先中断程序的正常执行，等待宿主机 GDB 的连接。此时，宿主机在 GDB 提示符下，运行 target remote 命令连接到目标机调试程序。若连接成功，开发人员就可以在主机使用 GDB 调试命令，对运行在目标系统中的程序进行远程调试了。

3.1 宿主机 GDB 的设置

宿主机环境的设置比较简单，只要有一个可

以运行 GDB 的系统环境，则大多数情况下选择一个较好的 Linux 发行版即可。但要注意，开发人员不能直接使用该发行版中的 GDB 来做远程调试，而要获取 GDB 的源文件包，针对特定目标平台做一个简单配置，重新编链得到相应的 GDB。笔者使用 gdb-5.2.1 源文件包，在解压后的目录中使用"./configure - - target = …；make"编链得到用于远程调试的宿主机 GDB。其中，configure 命令的 target 参数指定了目标机的类型，开发人员可从 GDB 说明文档获取 GDB 支持的目标平台类型。

3.2 目标机调试 stub 的一般实现

较之宿主机远程调试环境的建立，目标机调试 stub 的实现更复杂，它要提供一系列功能函数，以实现与主机 GDB 的通信和对被调试程序的控制。这些功能函数有的已由 GDB 提供，有的需要开发人员自行根据特定目标平台实现。如 GDB 文件包中的 m68k-stub.c、i386 - stub.c 等文件，提供了一些相应目标平台的 stub 子函数[3,4]：

set_debug_traps()：函数指针初始化，捕捉调试中断进入 handle_exception()函数。

handle_exception()：该函数是 stub 的核心部分。程序运行被中断时，首先发送一些主机的状态信息，如寄存器的值，然后在主机 GDB 的控制下执行程序，并检索和发送 GDB 需要的数据信息，直到主机 GDB 要求程序继续运行，handle_exception()交还控制权给程序。

breakpoint()：利用这个功能函数可以在被调试程序中设置断点。

除以上函数外，开发人员需要针对特定目标平台，为 stub 实现以下底层功能函数，才能使调试 stub 正常与主机 GDB 协同工作：

getDebugChar()、putDebugChar()：读写通过 GDB 远程串行协议与主机交互的数据。

exceptionHandler()：各目标平台对系统中断向量的组织安排是不一样的，该函数要能够使得系统中断发生时，程序可以正常获得中断服务程序的入口地址。

memset()：这是一个标准库函数，保证对特定目标平台的内存操作。

有了上述关键功能函数的实现，开发人员就可以按以下步骤[3]使用 stub 对目标程序进行远程调试：

(1) 在被调试程序开始处，插入两个函数调用：set_debug_traps()和 breakpoint()。

(2) 将被调试程序、GDB 提供的 stub 功能函数和上述目标系统中实现的 stub 底层子函数，一起编链生成一个包含调试 stub 的目标程序。

(3) 建立宿主机与目标机的串口或以太口连接，保证通信物理链路的通畅。

(4) 将被调试目标代码下载到目标系统，并运行该程序，它会被内部 stub 函数中断在开始处，等待宿主机 GDB 的连接请求。

(5) 在宿主机运行针对目标平台编链的 GDB，用 target remote 命令连接目标机 stub，然后就可以使用丰富的 GDB 命令对目标程序进行跟踪和调试了。

上述是设计和实现联调 stub 的一般方法，不同调试场合的 stub 的实现形式会有所不同，但使用不同形式 stub 对 Linux 内核和驱动模块进行调试的过程和操作则是大致相同的。

4 远程调试应用实例

目前而言，嵌入式 Linux 系统中，主要有三种远程调试方法，分别适用于不同场合的调试工作：用 ROM Monitor 调试内核装载程序、用 KGDB 调试系统内核和用 GDBserver 调试用户空间程序。这三种调试方法的区别主要在于，目标机远程调试 stub 的存在形式不同，而其设计思路和实现方法则大致是相同的，并且与它们配合工作的宿主机 GDB 是同一个程序，即 3.1 节所述。

4.1 用 ROM Monitor 调试目标机程序

嵌入式 Linux 系统在内核运行前的状态中，程序的装载、运行和调试一般都由 ROM Monitor 实现。系统一加电，包含了远程调试 stub 的 ROM Monitor 首先获得系统控制权，对 CPU、内存、中断、串口、网络等重要资源和外设进行初始化，然后就可以下载、运行和监控目标代码，包括内核的装载和引导也由它完成。基于 ARM 平台的嵌入式 Linux 系统中最常用的 RedBoot(Red Hat Embedded Debug and Bootstrap)[5]是一款功能相当强大的系统引导、调试和管理工具，其中包含了 GDB 远程调试 stub 的完整实现、串口和以太口的驱动及 Flash 设备的管理等功能。在内核运行之前，用 RedBoot 下载并调试 Linux 内核引导装载程序 armboot 的一个实例会话如下：

目标板加电，RedBoot 运行，在提示符下按 $ 键，进入调试 stub 状态：

```
ARM eCos // 部分 RedBoot 启动信息
RedBoot(tm) debug environment - built 15:49:04, Mar 27 2002
…… // 按 $ 键进入调试 stub 工作状态
RedBoot > Entering debug mode using GDB and stubs
宿主机运行 GDB，实现程序的下载和调试：
[root@Host armboot - 1.1.0] # gdb armboot
GNU gdb 5.2.1 // 部分 GDB 启动信息
Copyright 2002 Free Software Foundation, Inc. …
(gdb) set remotebaud 115200 // 设置串口波特率
(gdb) target remote /dev/ttyS0 // 通过串口 1 连接目标机
Remote debugging using /dev/ttyS0
0x0000e2b8 in ?? ()
(gdb) load // 下载目标代码至目标机内存
Loading section .text, size 0xbef8 lma 0xa3000000
Loading section .rodata, size 0x2f84 lma 0xa300bef8
Loading section .data, size 0xd24 lma 0xa300ee7c
Start address 0xa3000000, load size 64416
Transfer rate: 85888 bits/sec, 298 bytes/write.
(gdb)
```

至此，开发人员便可以使用 GDB 像调试桌面系统程序一样对目标程序进行跟踪调试了，用 list 察看代码，用 break 设置断点，然后用 continue 恢复程序的运行直至断点处。这时，我们就可以清晰地查看程序所使用的目标板资源的状态，如变量值、内存值、CPU 寄存器等等，这个场景将在 4.

2 节调试系统内核的会话中给出。

4.2 用 KGDB 调试系统内核

系统内核与硬件体系关系密切,因而其调试 stub 的实现也会因具体目标平台的差异而存在一些不同,嵌入式 Linux 开发团体针对大多数流行的目标平台,对 Linux 内核远程调试 stub 给予了实现,并以源码补丁形式发布。开发人员只需正确编链打好补丁的内核,就可对内核代码进行灵活的调试。PC 平台 Linux 内核开发人员所熟知的 KGDB(Remote Kernel Debugger,简称 KGDB)就是这种实现形式,该方法也同样用在嵌入式 Linux 系统中。如 MontaVista 的 Deepak Saxena 主持开发的内核调试补丁,就能够很好地完成 IOP3xx、ADI 和 IXP 系列目标平台上 Linux 内核的调试。通过串口以远程方式跟踪 do_fork()内核函数的大致场景如下:

```
(gdb) set remotebaud 115200 // 设置串口波特率
(gdb) target remote /dev/ttyS0 // 通过串口 1 连接远程目标
Remote debugging using /dev/ttyS0
……
(gdb) continue // 连接成功,开始内核的启动
Continuing.
Memory clock: 99.53MHz (*27) // 内核启动信息
Run Mode clock: 99.53MHz (*1)
……
VFS: Mounted root (nfs filesystem).
Freeing init memory: 76K
Program received signal SIGTRAP, // 中断内核的运行,以便设置断点
Trace/breakpoint trap.
……
(gdb) list fork.c:622 // 查看内核源代码
617 p->did_exec = 0;
618 p->swappable = 0;
619 p->state = TASK_UNINTERRUPTIBLE;
620
621 copy_flags(clone_flags, p);
622 p->pid = get_pid(clone_flags);
623
624 p->run_list.next = NULL;
625 p->run_list.prev = NULL;
626
(gdb) break 622 // 设置断点
Breakpoint 1 at 0xc0116d50: file fork.c, line 622.
(gdb) continue // 恢复内核的运行,直至断点
Continuing.
Breakpoint 1, do_fork (clone_flags=17, stack_start=3221223548, regs=0xc7153fc4, stack_size=0) at fork.c:622
622 p->pid = get_pid(clone_flags);
(gdb) display p->pid // 显示变量 p->pid 当前值
1: p->pid = 1152
(gdb) next 3 // 跟踪 get_pid()函数返回后,
624 p->run_list.next = NULL; // 变量 p->pid 的变化
1: p->pid = 1181
(gdb) delete 1 // 删除断点
(gdb) continue // 恢复内核的运行
Continuing.
```

GDB 的调试功能非常丰富和强大,上例所示只是 GDB 远程调试在系统内核调试中的一个最简单的应用,开发人员可以使用 GDB 提供的众多功能对内核进行强有力的跟踪调试。

4.3 用 GDBserver 调试用户空间程序

在 Linux 内核已经正常运行的基础上,使用 GDBserver 作为远程调试 stub 的实现,开发人员可以在宿主机上用 GDB 方便地监控目标机用户空间程序的运行。GDBserver 是 GDB 自带的、针对用户程序的远程调试 stub,它具有良好的可移植性,可交叉编译到多种目标平台上运行。因为有操作系统的支持,它的实现要比一般的调试 stub 简单很多,但作为以远程方式调试用户程序的目标方,正是其擅长之处。下面给出主机 gdb 和目标机 gdbserver 以 TCP 方式远程调试目标平台 MiniGUI 程序的大致过程[3],其中 210.195.150.140 是宿主机,210.195.150.164 为目标机。

```
目标机:
[root@l2x target]# gdbserver 210.195.150.140:2345 mginit
Process mginit created; pid=72 // 等待宿主机连接
宿主机:
[root@l2x host]# gdb mginit
…… // GDB 版本、配置信息
(gdb) target remote 210.195.150.164:2345 // TCP 方式连接目标机
Remote debugging using 210.195.150.164:2345
0x40002a90 in ?? () // 连接成功
目标机:
Remote debugging from host 210.195.150.140 // 宿主机连接成功
宿主机:
(gdb)… // 使用 GDB 命令进行调试过程
```

在开发嵌入式系统上层应用时,由于宿主机和目标机的软硬件环境存在或多或少的差异,所以开发人员在宿主机上模拟调试通过的程序往往并不能很好地运行于目标平台上。运用上述远程调试方法,则可以减少对模拟调试的依赖,直接跟踪目标平台上运行的程序,方便地找出程序臭虫,从而提高开发质量和效率,缩短嵌入式系统应用的开发周期。

在使用 GDB 对目标机程序进行远程调试时,首先要保证相关软件的正确设置和编链,如被调试程序是否包含了调试信息(编链选项 -g)、GDB 的版本和 target 等配置参数是否正确适合;其次要确认远程联调双方的物理链路(串口连接或网络连接)畅通。注意并解决好上述问题,双机远程联调的过程会更加令人愉快。

5 结束语

在嵌入式 Linux 系统开发中,GNU 工具链是一个很好的选择,它不但提供了对各种流行嵌入式处理器的良好支持,而且开发人员可免费使用,使得系统开发成本大大降低。GNU 工具链中的 GDB 更是以其远程的调试方式适应了嵌入系统的特殊调试要求,在实际开发中得到了最广泛的

(下转第 109 页)

究与发展,基于构件的软件开发已成为软件开发的一种潮流。如何迅速可靠地建立一个基于构件的信息系统,已成为一个重要的研究课题。本文提出了一个基于构件的信息系统开发构架,介绍了该框架中的组成元素及功能,将构件划分为业务构件和功能构件,并阐述了它们的设计方法。

参考文献:

[1] Somjit Arch-int, Dentcho N Batanov. Development of Industrial Information Systems on the Web Using Business Components[J]. Computers in Industry, 2003, 50(2): 231 - 250

[2] Ralph E Johnson. Frameworks Equal(Components + Patterns). Association for Computing Machinery[J]. Communications of the ACM, 1997, 40(10): 39 - 42.

[3] 吴明晖,应晶,何志均.基于构件的框架开发方法及其特定域应用[J].计算机工程,1999,25(10):86 - 92.

[4] M Shaw, D Garlan. Software Architecture: Perspectives on an Emerging Discipline[M]. Englewood Cliffs, NJ: Prentice Hall, Inc., 1996.

[5] Yang, Fu-qing. Software Reuse and Relevant Technology[J]. Computer Science, 1999, 26(5): 1 - 4.

(上接第 103 页)

应用。我们在一款 ARM 系列实验开发板上作嵌入式 Linux 系统开发时,使用本文所述的 GDB 远程调试方法,很好地完成了对内核装载程序、内核代码、用户空间程序的调试工作,系统开发效率得到了质的提高。

参考文献:

[1] 探矽工作室. 嵌入式系统开发圣经[M]. 沈阳: 中国青年出版社, 2002.

[2] 邹思轶. 嵌入式 Linux 设计与应用[M]. 北京: 清华大学出版社, 2002.

[3] Richard Stallman, Roland Pesch, Stan Shebs, et al. Debugging with GDB[EB/OL]. http://www.gnu.org/manual/gdb-5.1.1/html_mono/gdb.html, 2002 - 01.

[4] Bill Gatliff. Embedding with GNU: The GNU Debugger[EB/OL]. http://www.redhat.com/devnet/articles/embedgdb.htm, 2001 - 02.

[5] Red Hat Inc. RedBoot(tm) User's Guide[EB/OL]. http://sources.redhat.com/ecos/docs-latest/redboot/redboot.html, 2001 - 07.

[6] Anthony J Massa. If the RedBoot Fits[EB/OL]. http://www.embedded.com/story/OEG20020729S0043, 2002 - 07.

(上接第 105 页)

估结果如表 1 所示。这说明专家系统对银行贷款风险评估已接近于若干专家根据经验所进行的评估,初步达到了预期的研究目的。

表 1 专家系统的验证与结果

评估结果	正常贷款		逾期贷款		呆滞贷款		呆帐贷款	
	户数	比例	户数	比例	户数	比例	户数	比例
1~3级(同意授信)	91	91%	11	11%	1	2%	0	0%
5~7级(考虑交易条款和综合创利)	7	7%	72	72%	5	10%	4	8%
8~10级(拒绝授信)	2	2%	17	17%	44	88%	46	92%

4 结束语

本系统已于 2001 年在某商业银行营业部推广应用,并取得了准确、高效、通用性好等显著效果。结合现代商业银行扁平化管理体系及客户经理制的推广实行,该商业银行授信流程精简为四级,一笔贷款的审批时间由以前的 10~15 天减少到 5~7 天。自 2001 年以来,该商业银行新放贷款本息回收率达 100%,且不良贷款率比上期明显下降。此外,通过对知识库的修改,本系统不仅适用于多家商业银行,而且同样适用于一家商业银行的不同发展时期。

现阶段的银行贷款风险评估专家系统还处于初级阶段,存在的主要问题是专家系统的知识源和知识学习问题。因此,探索新的贷款风险评估技术和理论方法,研究开发完善的智能贷款风险评估系统是今后发展的主要方向。

参考文献:

[1] 张亚红.试论我国潜在的金融风险与防范对策[J].决策借鉴,1999,(3):10 - 13.

[2] 郭慧.专家系统的一种智能知识获取技术研究[J].化工装备技术,1999,(2):15 - 27.

[3] 胡守仁.神经网络导论[M].长沙:国防科技大学出版社,1993.

[4] Fu L M. Learning Capacity and Sample Complexity on Expert Networks[J]. IEEE Trans on Neural Networks, 1996, (11): 33 - 37.