GDBSERVER 原理分析及其应用

陈必泉， 黄承慧
(暨南大学 计算机科学与工程系，广东 广州 510632)

摘 要: GDBSERVER是一个轻量级的运行于目标机上的调试器，在嵌入式Linux系统开发中发挥着重要作用。从源代码层分析了GDBSERVER的实现原理，并介绍了在嵌入式Linux系统开发中使用GDBSERVER进行远程调试的方法。

关键词: gdbserver; 调试代理; 远程调试; 嵌入式系统
中图法分类号: TP311.6 **文献标识码**: A **文章编号**: 1000-7024 (2005) 03-0746-04

Analysis of GDBSERVER and its application

CHEN Bi-quan, HUANG Cheng-hui
(Department of Computer Science and Technology, Jinan Univesity, Guangzhou 510632, China)

Abstract: GDBSERVER is a lightweight debugger which runs on target, and plays an important role on the development of embedded Linux system. The implementation of GDBSERVER is analyzed from the source code, and the method of remotely debugging is introduced using GDBSERVER in the development of embedded Linux system.

Key words: gdbserver; debugging agent; remote debugging; embedded system

1 引 言

当前，在嵌入式处理器领域ARM系列的32位嵌入式芯片，以其耗电少、成本低、功能强、特有的16/32位双指令集，已成为移动通信、手持设备、多媒体数字消费等嵌入式解决方案的RISC标准。同时，一些优秀的操作系统被移植到ARM系统中来，如Linux。与其它嵌入式操作系统相比，Linux以其易于移植、效率高、强大的网络功能、代码开放等诸多优势，在嵌入式领域得到广泛认同。基于Linux进行开发不但可以使开发人员专注于嵌入式应用软件的开发，而且还使得开发人员很容易获得免费的功能强大的开发和调试工具，从而能降低成本缩短研发周期。

嵌入式系统开发中，使用的是交叉开发模型：软件开发和部分测试工作在主机上完成，最终的软件产品要在目标机上运行。由于开发平台和运行平台的不一致性，嵌入式系统所控制的外部设备的复杂性、可靠性及实时性要求等诸多因素使得嵌入式软件调试非常复杂。目前，嵌入式系统的调试方法主要有：实时在线仿真器(ICE)、源程序模拟器、线上调试技术、使用调试代理(debugging agent)等。实时在线仿真器是功能最强大的软件调试工具之一，但价格昂贵缺乏通用性；源程序模拟器提供的运行环境在许多方面与真实环境有较大差距，因而影响调试效果。线上调试技术利用了微处理器内部的仿真功能，为高档嵌入式系统开发提供各种非干扰的调试手段。与前面的方法相比，调试代理则提供了一种价低且具有一定通用性的调试手段。在嵌入式Linux开发领域里，常用的调试代理工具为GDBSERVER。它是一个轻量级的GDB调试器，运行在目标机上，与运行在主机上的GDB通过RSP(Remote Serial Protocol)协议进行通讯从而完成远程调试工作。

2 GDB/GDBSERVER 调试技术基础

2.1 GDB/GDBSERVER 调试模型

调试是软件开发中的重要的一环，它占整个嵌入式系统开发时间的20%-50%。在调试过程中，用户在调试环境的监视下分析所要调试的程序，发现程序中存在的问题与缺陷，从而达到修正程序的目的。实践证明，一个好的调试方法可以有效帮助程序员查找程序中的错误，从而减少软件开发的时间。在面向桌面Linux系统的软件开发过程中，可以使用GNU提供的工具 GDB 来对编译好的程序进行本地调试。在嵌入式Linux开发中，软件的开发平台和运行平台是不一致的，因此要在目标机上进行调试，除了在主机上运行的GDB外，还需要一个运行在目标机上的 GDBSERVER 调试器。GDBSERVER是一个轻量级的GDB调试器，它在调试过程中担任着调试代理的角色。

在调试过程中，主机和目标机之间使用串口或者网络作为通信的通道，如图1所示。在主机上GDB通过这条通道使用一种基于ASCII的简单通讯协议RSP与在目标机上运行的GDBSERVER进行通讯。GDB发送调试指令，如设置断点、步进、内存/寄存器读写。GDBSERVER首先要与运行被调试程

收稿日期: 2004-06-25。
作者简介: 陈必泉 (1979-)，男，广西柳州人，硕士生，研究方向为嵌入式软件测试； 黄承慧 (1976-)，男，湖南永兴人，硕士生，研究方向为面向对象的软件测试。

序映像的进程进行绑定，然后等待 GDB 发来的数据。在对包含命令的数据进行解析之后便进行相关处理，然后将结果返回给主机上的 GDB。

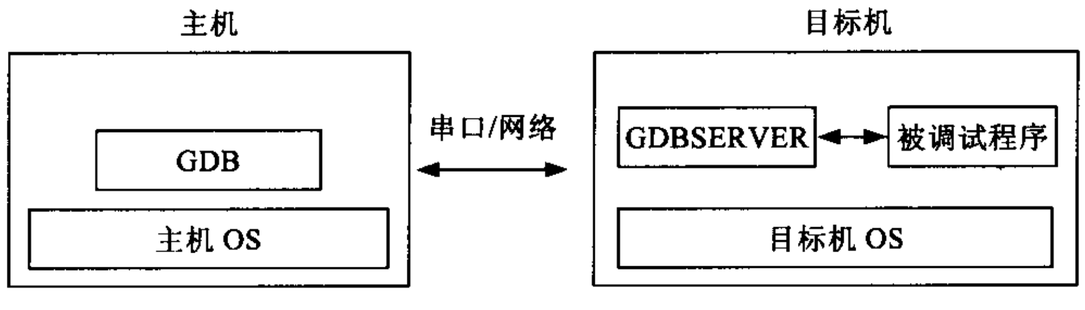

图 1　GDB/GDBSERVER 调试模型

2.2　RSP 通讯协议

RSP 协议将 GDB/GDBSERVER 间通讯的内容看做是数据包，数据包的内容都使用 ASCII 字符。每一个数据包都遵循这样的格式：$ <调试信息>#<校验码>.接受方在收到数据包之后，对数据包进行校验，若正确回应“+”，反之回应“-”。

RSP 协议中定义的主要命令可以分为 3 类：

(1) 寄存器/内存读写命令

命令 g: 读所有寄存器的值

命令 G: 写所有寄存器的值

命令 P: 写某个寄存器

命令 m: 读某个内存单元

命令 M: 写某个内存单元

(2) 程序控制命令

命令?: 报告上一次的信号

命令 s: 单步执行

命令 c: 继续执行

命令 k: 终止程序

(3) 其它命令

命令 O: 控制台输出(Console Output)

命令 E: 出错回应(Error response)

3　GDBSERVER 原理分析

GDBSERVER 执行的任务主要有：进程绑定和调试数据的解析及相应处理操作的执行。要调试目标机上的应用程序，首先要让 GDBSERVER 对被调试程序的进程进行绑定。所谓绑定就是指调试进程让被调试的应用程序的进程成为自己的子进程，这样调试进程就可以利用内核提供的代码跟踪机制来完成调试任务。绑定的工作是由函数 start_inferior()来完成。在绑定之后，就进入一个 while 循环体。循环体开始处，程序首先调用函数 remote_open()函数来打开同主机通讯的端口，之后调用 getpkt()函数获取从主机发过来的调试数据，并存放在缓冲区 own_buf 中。在获得有效的调试数据后，GDBSERVER 使用一个 switch/case 结构对数据进行解析(如下面的伪代码)。

```
SWITCH (COMMAND)
{   ......
  CASE 'g':          /* 如果是命令 g
    ReadAllRegisters() /* 则读取所有寄存器的值到缓冲区中
  CASE 'G':          /* 如果是命令 G
    WriteAllRegisters()/* 则将指定的值写入到寄存器中
  ......
  DEFAULT:
    UnknownCmd()     /* 处理未定义命令
}
```

如果当前的调试数据是有效的调试命令，那相关的处理操作就会与之匹配从而被执行。switch/case 结构提供了一个清晰的构架同时也给处理 RSP 协议的扩展问题预留了空间。

3.1　进程绑定

start_inferior()函数用来对进程进行绑定，存在两种情况。

(1) 如果需要调试的程序已经运行则使用函数 attach_inferior()完成绑定，其过程如下：获取已运行程序对应的进程号然后将该进程收养为子进程，即将子进程的 task 结构中的 p_pptr 项设为当前 GDBSERVER 对应的进程，然后向该子进程发送 SIGSTOP 信号。在 linux 内核中 SIGSTOP 和 SIGKILL 这两个信号不能被屏蔽且对它们的响应操作也不能改变。GDBSERVER 正是利用了信号机制来实现调试进程和被调试进程之间的同步。当被绑定进程收到 SIGSTOP 信号后，就会暂停下来，进入 TASK_STOPED 状态。GDBSERVER 的进程发送完信号后不会马上返回，而是调用一个等待函数。只有当子进程停止了运行且准备好接受调试时，GDBSERVER 进程才继续执行。在这种情况下有 3 种限制：不许自我绑定，不许多次绑定同一个进程，不能绑定 1 号进程。

(2) 如果需要调试的程序没有运行则调用 create_inferior()函数。这个函数会创建一个子进程。子进程执行后，调用 ptrace()并设置 PTRACE_TRACEME 标志，然后调用 execv ()函数执行要调试的程序映像。ptrace()是类 unix 系统中一个用来调试的系统调用，调试过程中的大部分工作都是使用该调用来完成，该调用的原理将在 3.2 节进行介绍。由于在调用 ptrace()时使用了参数 PTRACE_TRACEME，子进程会停在被调试程序的第 1 个指令处，并向父进程发送消息，等待调试。

3.2　PTRACE () 调用分析

ptrace()是 Linux 内核提供的一个用于进程跟踪的系统调用。通过它，一个进程可以读写另外一个进程的指令空间、数据空间、堆栈和该进程运行时所有寄存器的值，通过与信号机制相结合可以实现让一个进程在另一个进程的控制和跟踪下运行。GDBSERVER 正是使用了这一功能来完成所有的调试工作。

ptrace()的调用接口原型为 int ptrace(int request, int pid, int addr,int *data)，其中 request 为操作请求，其取值范围为内核中定义的所有操作码，pid 为进程号，addr 为子进程地址空间中的某一地址，*data 表示需要从 addr 读入或者要写到 addr 处的数据。调试进程使用这个调用接口来指定操作内容。2.4.x 的内核代码中，ptrace.h 文件定义了一系列的操作码，表 1 中列出了主要的几项。

与其它系统调用一样，ptrace () 在内核中的实现是 sys_ptrace()，其代码定义在 arch 目录下。arch 下有若干子目录，每一个子目录代表一种 cpu 和体系结构，其下存放内核中与体系结构相关的代码。本小节的分析针对 arm 系列 cpu 的实现代码。sys_ptrace()流程如下：首先会处理 PTRACE_TRACEME 请求。该请求仅由被调试进程在调用 ptrace()时使用，表示主

表 1　ptrace 操作码

操作码	操作描述
PTRACE_TRACEME	设置当前进程的 PF_PTRACED 标志，表示当前进程主动接受跟踪
PTRACE_ATTACH	绑定被跟踪进程，发送 SIGSTOP 信号使被跟踪进程暂停
PTRACE_PEEKDATA	从子进程数据空间读取一个长字的数据
PTRACE_PEEKUSR	读取子进程在用户空间运行时某个寄存器的值
PTRACE_POKEDATA	向子进程数据空间写一个长字的数据
PTRACE_POKEUSR	写子进程在用户空间运行时某个寄存器的值
PTRACE_GETREGS	获取所有 gp 寄存器的值
PTRACE_SYSCALL	设置 PF_TRACESYS 标志，清除 TRAP_FLAG 标志，使得子进程在下一次系统调用时继续或停止
PTRACE_CONT	下一次信号重启
PTRACE_KILL	使子进程退出运行
PTRACE_SINGLESTEP	执行一条指令

动接受跟踪。如果是 PTRACE_ATTACH 请求则调用 ptrace_attach()来完成对进程的绑定操作。通过分析可以得知前两个操作码对应的操作用来完成调试前的准备工作。对其它的操作码的处理通过调用 do_ptrace()函数来完成。这些操作码大体可以分为两类：一类用于检查和设置如PTRACE_PEEKDATA、PTRACE_GETREGS；另一类用于控制进程执行如 PTRACE_SYSCALL、PTRACE_SINGLESTEP 等。在这种情况下，对子进程相应设置完成后，wake_up_process () 会被执行以唤醒子进程。

4　GDBSERVER 在嵌入式开发中的应用

在嵌入式 Linux 开发中使用 GDBSERVER 进行调试，可以分为 4 个步骤：建立交叉编译环境，安装GDBSERVER，建立远程调试环境，调试程序。

4.1　建立交叉编译环境

由于主机和目标机的cpu不同，所以编译在目标机上运行程序时必须使用交叉工具包。GNU 交叉工具包包括以下几个部分：专用平台编译器和 binnutils。后者是一些辅助工具，包括可以反编译二进制文件的工具(objdump)、汇编编译器(as)、连接器(ld)等。这些工具的安装既可以通过下载源代码编译后安装，也可以通过下载相应的二进制 rpm 文件，使用 rpm 命令来安装。针对 arm 系列的 cpu，相应工具的二进制文件是：

arm-elf-binutils-v.i386.rpm

arm-elf-gcc-v.i386.rpm

这里的 v 代表版本号。此外还可以安装嵌入式 Linux 系统的 C 库 uClibc。该库比 GNU C 库更小，但是 glibc 支持的所有应用软件几乎都可以使用它。将应用软件从 glibc 移植到 uClibc 库，通常只需要重新编译源代码。uClibc 支持标准的 Linux 系统(像 x86、strongArm 以及 powerpc)，此外，它还支持无 MMU 的系统(uClinux)，像基于 Coldfire、dragonball 以及 arm7 的微控制器。利用 uClibc 构建嵌入式 Linux 系统将比 glibc 占用更小的空间。

4.2　安装 GDBSERVER

我们使用了优龙公司的基于arm的4510开发板和开发套件。开发套件中包含了 GDBSERVER 的源代码，因此，只需在使用开发包打造嵌入式 Linux 操作系统时，在配置栏中选中 GDBSERVER 的选项，编译好的文件系统中就会包含 GDBSERVER 程序。此外也可以下载 GDBSERVER 的源代码在主机上单独编译，然后通过 FTP 工具将编译得到可执行文件下载到目标机上或者用 4.3 节介绍的 NFS 方法来执行该程序。GDBSERVER 的源代码被包含在了 GDB 源代码的 GDBSERVER目录下，GDB5.1.1 以后的版本都加入了对ARM处理器的支持。在编译之前，需要对源代码中的 Makefile 文件做一定修改，如指定编译器为 arm-elf-gcc、使用 uClibc 库。

4.3　建立远程调试环境

调试程序前需要解决两个问题：目标机的监控和被调试程序的加载。

(1) 目标机的监控

在目标机上 GDBSERVER 一般通过手工执行，有时还需要在目标机系统上执行一些相关操作。而目标机上通常没有显示器、键盘之类的显示和输入设备，这就需要在主机环境下能够提供一种可以远程控制目标机的手段。在 Linux 环境下，可以使用 minicom 程序通过串口登陆到目标机上。minicom 是一个通信终端程序，通过 minicom 可以设置、监视串口工作状态，接收、显示串口收到的信息，并且在主机和目标机之间传递数据和控制指令，从而实现通过主机监控目标机的目的。

(2) 被调试程序的加载

调试时，GDBSERVER 生成的子进程需要加载被调试程序，因此被调试程序必须存在于目标机的文件系统当中。加载被调试程序到目标机文件系统有多种方法，比如被调试程序可以编译到文件系统的映像文件里，然后通过烧写工具写到目标机的 flashrom 里。但是对程序的每次修改都要重新编译和烧写。在调试阶段更常用的方法是使用 ftp 工具和 NFS 文件系统。前者只需要在为目标机定制嵌入式 Linux 操作系统时将 ftp 工具包含到文件系统中，再将主机配置成 ftp 服务器，将程序从主机下载到目标机上即可。后者是 Network File System 的缩写，它提供了一种在不同机器之间共享文件的办法。其基本思想是某一台 Linux 主机作为 NFS 服务器输出其文件系统，其它机器获得授权将该文件系统加载自己的主文件系统的某一目录下。

使用 NFS 需要对主机和目标机做相应的配置。在主机上需要编辑/etc/exports。该文件列出了需要输出的文件系统和输出方式，其格式为：directory hostname(options)。其中 hostame 是主机名，也可以是 ip 地址，它指明了文件系统的输出对象。假设目标机的 ip 为 192.168.0.2，用户程序在/usr 目录下，则主机上/etc/exports 的内容为：

/usr　　192.168.0.2(ro)

编辑完后，可以通过下面的命令使新的设置生效：

/etc/rc.d/init.d/nfs restart

为了让目标机上的 linux 支持 nfs，在配置嵌入式 linux 内核时要加入以下两项：NFS file system support 和 Provide NFSv3 client support。此外最后得到的文件系统中还应该包含 mount 和 portmap 两个程序。前者用来加载 NFS 输出的文件系统，后者负责端口映射(port mapping)。

4.4 调试程序

目标机加电运行后，可以使用minicom通过串口连接到目标机上，并以后台运行的方式执行 portmap 命令。假设主机上的输出目录为/usr,目标机上的加载点为/var, 需要调试的程序为 myprogram，存放在主机的/usr/debug 下，那么使用如下命令安装 nfs:

mount 主机 ip:/usr /var

安装好 nfs 后，使用如下命令启动 GDBSERVER:

gdbserver :PORTNUM /var/usr/debug/myprogram

其中 PORTNUM 是一个没有被使用的端口号。这时候 GDBSERVER 处于就绪状态，等待主机上的 GDB 发来的调试指令。之后在主机上启动GDB，出现GDB提示符后输入命令：

remote 目标机 ip:PORTNUM

就可以使用 GDB 提供的一系列的调试命令如断点、单步执行、监视变量等来对应用程序进行调试。

5 结束语

GDBSERVER 提供了一种方便有效的调试嵌入式应用程序的手段，在实际的开发过程中发挥着重要作用。但是由于其进程控制的基础是 ptrace()调用，因而有其局限性。比如只能跟踪它的子进程，在调试进程和被调试进程之间传送一个长字的数据，内核必须完成 4 次上下文的切换。针对 ptrace()的不足，killian 提出了基于进程文件系统/proc 的跟踪进程方案。在该方案下，调试进程可以通过普通的读写函数来访问被调试进程的地址空间和设置断点，而且克服了前面提到的ptrace()的局限性。主机上运行的 GDB 就采用了这种方案。但是，尽管如此，ptrace()作用还是不能被完全取代。

参考文献:

[1] 李红卫, 李翠萍. 嵌入式软件的调试技术[J]. 计算机时代, 2002, (8):3-4.

[2] 张卫民, 黄瑞芳, 张钦伍. 基于进程文件系统的调试器设计[J]. 小型计算机系统, 1996, 17(2):41-46.

[3] 张栋岭, 刘献科, 邓晓艳, 等. 嵌入式应用的远程调试[J]. 计算机工程, 2003, 29(11):76-78.

[4] 毛德操, 胡希明. LINUX 内核源代码情景分析[M]. 杭州:浙江大学出版社, 2001.

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

[6] 李杰, 贺占庄. 在 MIPS 开发板上建立 Linux 系统及开发环境[EB/OL]. 2003-11-26. http://www.21ic.com/new_info/news/files/news/2003112611402.asp.

[7] BILL GATLIFF. Embedding with GNU:GNU debugger[J]. EmbeddedSystem Programming, 1999, (9):80-94.

[8] BILL GATLIFF. Embedding with GNU:The gdb remote serial protocol [J]. Embedded System Programming, 1999, (11):108-113.

(上接第 687 页)

3.4 远程监测模块

当设备出现重大或疑难问题，为了对其进行快速有效的排故，结合维修和排故需要，通过远程网络而采取的一种对异地设备进行的系统级或部件级的状态监测，最终达到对系统或部件的远程检测、故障诊断和故障辨识。其主要手段包括对设备的远程测试(测试其不同工作状态下的性能参数)、远程控制、特征分析(通过现场端的数据采集设备采集设备的特征参数)等。设备远程监测使得对设备的测试模式由过去功能单一、地域集中型向功能综合、地域分布型和网络化分布式集成应用型方式转变。

远程监测使得专家可以从异地通过网络对现场设备进行监控和测试，在远端再现故障现场的设备状态，从而辅助现场人员进行维修排故，同时将监测数据存放在诊断中心的历史数据库中，供设备离线时进行分析。

4 原型系统的设计开发

基于前文提出的总体结构和功能模型，我们开发出了一个基于 Web 的远程故障诊断平台的原形系统，并针对具体的诊断对象和采集设备开发出了相应的组件和控件。

诊断中心的Web服务器和数据库服务器的操作系统使用 Microsoft Windows 2000 Advanced Server，采用 IIS 5.0 作为 Web 服务器，SQL Server 2000 作为后台数据库系统，采用功能强大的 ADO(Microsoft ActiveX Data Objects)作为 Web 与数据库之间的接口技术。Web 页面部分的代码用 HTML 和 ASP 编写，并在部分地方使用了ASP.Net；服务器端组件和嵌入页面的ActiveX 控件采用 Delphi 和 Visual C++编写。

5 结束语

基于Web的远程诊断为设备故障诊断的远程化、网络化、多专家协同诊断以及诊断资源的共享提供了一个便捷的途径，大大提高设备故障诊断的工作效率，并且它实现了开发环境和应用环境的分离，使开发环境独立于用户的应用环境，便于系统的维护、扩展和管理。

本文结合原型系统的开发和应用，证明了这一技术是可行的和有效的。

参考文献:

[1] 曾锋. 远程设备故障诊断技术研究[D]. 郑州:郑州大学, 2002.

[2] 楼佩煌, 戴勇. 基于 Internet 的远程故障诊断与维修向导系统研究[J]. 中国机械工程, 2002, 13(2):143-146.

[3] 张天宏, 左江福. 民航发动机远程故障诊断若干关键技术研究[J]. 航空动力学报, 2003, 18(1):33-37.

[4] 王长琼. 基于 Web 模式和虚拟现实的远程诊断技术研究[J]. 武汉理工大学学报, 2002, 26(1):66-67.

[5] 尹洪珠. 基于 Internet 的机械设备的远程故障诊断中若干问题的研究[D]. 大连: 大连理工大学, 2002.

[6] 侯作猛. 基于 Internet 的机械设备远程故障诊断平台[D]. 武汉: 武汉理工大学, 2001.