Linux 系统内核的调试

调试是软件开发过程中一个必不可少的环节，在 Linux 内核开发的过程中也不可避免地会面对如何调试内核的问题。但是，Linux 系统的开发者出于保证内核代码正确性的考虑，不愿意在 Linux 内核源代码树中加入一个调试器。他们认为内核中的调试器会误导开发者，从而引入不良的修正[1]。所以对 Linux 内核进行调试一直是个令内核程序员感到棘手的问题，调试工作的艰苦性是内核级的开发区别于用户级开发的一个显著特点。

尽管缺乏一种内置的调试内核的有效方法，但是 Linux 系统在内核发展的过程中也逐渐形成了一些监视内核代码和错误跟踪的技术。同时，许多的补丁程序应运而生，它们为标准内核附加了内核调试的支持。尽管这些补丁有些并不被 Linux 官方组织认可，但他们确实功能完善，十分强大。调试内核问题时，利用这些工具与方法跟踪内核执行情况，并查看其内存和数据结构将是非常有用的。

本文将首先介绍 Linux 内核上的一些内核代码监视和错误跟踪技术，这些调试和跟踪方法因所要求的使用环境和使用方法而各有不同，然后重点介绍三种 Linux 内核的源代码级的调试方法。

1. Linux 系统内核级软件的调试技术

printk() 是调试内核代码时最常用的一种技术。在内核代码中的特定位置加入printk() 调试调用，可以直接把所关心的信息打打印到屏幕上，从而可以观察程序的执行路径和所关心的变量、指针等信息。 Linux 内核调试器（Linux kernel debugger，kdb）是 Linux 内核的补丁，它提供了一种在系统能运行时对内核内存和数据结构进行检查的办法。Oops、KDB在文章掌握 Linux 调试技术有详细介绍，大家可以参考。 Kprobes 提供了一个强行进入任何内核例程，并从中断处理器无干扰地收集信息的接口。使用 Kprobes 可以轻松地收集处理器寄存器和全局数据结构等调试信息，而无需对Linux内核频繁编译和启动，具体使用方法，请参考使用 Kprobes 调试内核。

以上介绍了进行Linux内核调试和跟踪时的常用技术和方法。当然，内核调试与跟踪的方法还不止以上提到的这些。这些调试技术的一个共同的特点在于，他们都不能提供源代码级的有效的内核调试手段，有些只能称之为错误跟踪技术，因此这些方法都只能提供有限的调试能力。下面将介绍三种实用的源代码级的内核调试方法。

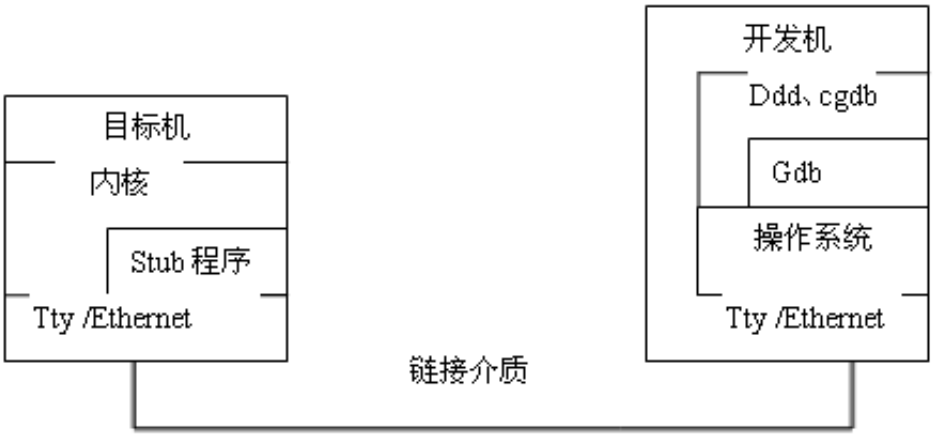
2. 使用KGDB构建Linux内核调试环境

kgdb提供了一种使用 gdb调试 Linux 内核的机制。使用KGDB可以象调试普通的应用程序那样，在内核中进行设置断点、检查变量值、单步跟踪程序运行等操作。使用KGDB调试时需要两台机器，一台作为开发机（Development Machine）,另一台作为目标机（Target Machine），两台机器之间通过串口或者以太网口相连。串口连接线是一根RS-232接口的电缆，在其内部两端的第2脚（TXD）与第3脚（RXD）交叉相连，第7脚（接地脚）直接相连。调试过程中，被调试的内核运行在目标机上，gdb调试器运行在开发机上。

目前，kgdb发布支持i386、x86_64、32-bit PPC、SPARC等几种体系结构的调试器。有关kgdb补丁的下载地址见参考资料[4]。

2．1 kgdb的调试原理

安装kgdb调试环境需要为Linux内核应用kgdb补丁，补丁实现的gdb远程调试所需要的功能包括命令处理、陷阱处理及串口通讯3个主要的部分。kgdb补丁的主要作用是在Linux内核中添加了一个调试Stub。调试Stub是Linux内核中的一小段代码，提供了运行gdb的开发机和所调试内核之间的一个媒介。gdb和调试stub之间通过gdb串行协议进行通讯。gdb串行协议是一种基于消息的ASCII码协议，包含了各种调试命令。当设置断点时，kgdb负责在设置断点的指令前增加一条trap指令，当执行到断点时控制权就转移到调试stub中去。此时，调试stub的任务就是使用远程串行通信协议将当前环境传送给gdb，然后从gdb处接受命令。gdb命令告诉stub下一步该做什么，当stub收到继续执行的命令时，将恢复程序的运行环境，把对CPU的控制权重新交还给内核。



2．2 Kgdb的安装与设置

下面我们将以Linux 2.6.7内核为例详细介绍kgdb调试环境的建立过程。

2.2.1软硬件准备

以下软硬件配置取自笔者进行试验的系统配置情况：

硬件	目标机	开发机	备注
IP 地址	192. 168. 5. 13	192. 168. 6. 13	—
连接端口	Com1	Com1	试验选用串口连接，调试端口的使用可选用以太网口
操作系统	Fedora 3	Fedora 3	选用 redhat 7.3 或以后版本
Linux 内核	Linux 2.6.7		—
kgdb 内核补丁	linux-2.6.7-kgdb-2.2.tar.tar		下载与 Linux 内核版本相对应的 kgdb 补丁。
串口线	使用空调制解调器电缆		或者按要求自己制作串口电缆

表 1：系统软硬件配置表

kgdb补丁的版本遵循如下命名模式：Linux-A-kgdb-B，其中A表示Linux的内核版本号，B为kgdb的版本号。以试验使用的kgdb补丁为例，linux内核的版本为linux-2.6.7，补丁版本为kgdb-2.2。

物理连接好串口线后，使用以下命令来测试两台机器之间串口连接情况，stty命令可以对串口参数进行设置：

在development机上执行：

```
stty ispeed 115200 ospeed 115200 -F /dev/ttyS0
```

在target机上执行：

```
stty ispeed 115200 ospeed 115200 -F /dev/ttyS0
```

在developement机上执行：

```
echo hello > /dev/ttyS0
```

在target机上执行：

```
cat /dev/ttyS0
```

如果串口连接没问题的话在将在target机的屏幕上显示"hello"。

2．2．2 安装与配置

下面我们需要应用kgdb补丁到Linux内核，设置内核选项并编译内核。这方面的资料相对较少，笔者这里给出详细的介绍。下面的工作在开发机（developement）上进行，以上面介绍的试验环境为例，某些具体步骤在实际的环境中可能要做适当的改动：

I、内核的配置与编译

```
[root@lisl tmp]# tar -jxvf linux-2.6.7.tar.bz2
[root@lisl tmp]#tar -jxvf linux-2.6.7-kgdb-2.2.tar.tar
[root@lisl tmp]#cd inux-2.6.7
```

请参照目录补丁包中文件README给出的说明，执行对应体系结构的补丁程序。由于试验在i386体系结构上完成，所以只需要安装一下补丁：core-lite.patch、i386-lite.patch、8250.patch、eth.patch、core.patch、i386.patch。应用补丁文件时，请遵循kgdb软件包内series文件所指定的顺序，否则可能会带来预想不到的问题。eth.patch文件是选择以太网口作为调试的连接端口时需要运用的补丁

。

应用补丁的命令如下所示：

```
[root@lisl tmp]#patch -p1 <../linux-2.6.7-kgdb-2.2/core-lite.patch
```

如果内核正确，那么应用补丁时应该不会出现任何问题（不会产生*.rej文件）。为Linux内核添加了补丁之后，需要进行内核的配置。内核的配置可以按照你的习惯选择配置Linux内核的任意一种方式。

```
[root@lisl tmp]#make menuconfig
```

在内核配置菜单的Kernel hacking选项中选择kgdb调试项，例如：

```
  [*] KGDB: kernel debugging with remote gdb
       Method for KGDB communication (KGDB: On generic serial port (8250))  --->
  [*] KGDB: Thread analysis
  [*] KGDB: Console messages through gdb
[root@lisl tmp]#make
```

编译内核之前请注意Linux目录下Makefile中的优化选项，默认的Linux内核的编译都以-O2的优化级别进行。在这个优化级别之下，编译器要对内核中的某些代码的执行顺序进行改动，所以在调试时会出现程序运行与代码顺序不一致的情况。可以把Makefile中的-O2选项改为-O,但不可去掉-O，否则编译会出问题。为了使编译后的内核带有调试信息，注意在编译内核的时候需要加上-g选项。

不过，当选择"Kernel debugging->Compile the kernel with debug info"选项后配置系统将自动打开调试选项。另外，选择"kernel debugging with remote gdb"后，配置系统将自动打开"Compile the kernel with debug info"选项。

内核编译完成后，使用scp命令进行将相关文件拷贝到target机上(当然也可以使用其它的网络工具，如rcp)。

```
[root@lisl tmp]#scp arch/i386/boot/bzImage root@192.168.6.13:/boot/vmlinuz-2.6.7-kgdb
[root@lisl tmp]#scp System.map root@192.168.6.13:/boot/System.map-2.6.7-kgdb
```

如果系统启动使所需要的某些设备驱动没有编译进内核的情况下，那么还需要执行如下操作：

```
[root@lisl tmp]#mkinitrd /boot/initrd-2.6.7-kgdb 2.6.7
[root@lisl tmp]#scp initrd-2.6.7-kgdb root@192.168.6.13:/boot/ initrd-2.6.7-kgdb
```

II、kgdb的启动

在将编译出的内核拷贝的到target机器之后，需要配置系统引导程序，加入内核的启动选项。以下是kgdb内核引导参数的说明：

kgdboe=@local-ip/,@remote-ip/	当使用网络接口作为调试作为启动参数告诉 stub，指定 target 机和 develop 机的 IP 地址。
2.0 版本以前的 kgdb： gdb gdbttyS=1 gdbbaud=115200	
启动参数	含义
Gdb	内核启动时等待 gdb 连接
gdbttyS	该选项指定 kgdb stub 使用哪一个串口进行通讯。取值为 0－3 分别代表 ttyS0 到 ttyS3 端口。
Gdbbaud	指定串口的波特率，波特率范围为 9600 to 115200
2.0 版本以后的 kgdb： kgdbwait kgdb8250=0,115200	
Kgdbwait	内核启动时等待 gdb 连接
kgdb8250=<port number>,<port speed>	该选项指定 kgdb stub 使用哪一个串口进行通讯，取值为 0－3。并指定端口的波特率，所支持的波特率为 9600, 19200, 38400, 57600 and 115200。

表 2：kgdb 内核引导参数

如表中所述，在kgdb 2.0版本之后内核的引导参数已经与以前的版本有所不同。使用grub引导程序时，直接将kgdb参数作为内核vmlinuz的引导参数。下面给出引导器的配置示例。

```
title 2.6.7 kgdb
root (hd0,0)
kernel /boot/vmlinuz-2.6.7-kgdb ro root=/dev/hda1 kgdbwait kgdb8250=1,115200
```

在使用lilo作为引导程序时，需要把kgdb参放在由append修饰的语句中。下面给出使用lilo作为引导器时的配置示例。

```
image=/boot/vmlinuz-2.6.7-kgdb
label=kgdb
    read-only
    root=/dev/hda3
append="gdb gdbttyS=1 gdbbaud=115200"
```

保存好以上配置后重新启动计算机，选择启动带调试信息的内核，内核将在短暂的运行后在创建init内核线程之前停下来，打印出以下信息，并等待开发机的连接。

Waiting for connection from remote gdb...

在开发机上执行：

```
gdb
file vmlinux
set remotebaud 115200
target remote /dev/ttyS0
```

其中vmlinux是指向源代码目录下编译出来的Linux内核文件的链接，它是没有经过压缩的内

核文件，gdb程序从该文件中得到各种符号地址信息。

这样，就与目标机上的kgdb调试接口建立了联系。一旦建立联接之后，对Linux内的调试工作与对普通的运用程序的调试就没有什么区别了。任何时候都可以通过键入ctrl+c打断目标机的执行，进行具体的调试工作。

在kgdb 2.0之前的版本中，编译内核后在arch/i386/kernel目录下还会生成可执行文件gdbstart。将该文件拷贝到target机器的/boot目录下，此时无需更改内核的启动配置文件，直接使用命令：

```
[root@lisl boot]#gdbstart -s 115200 -t /dev/ttyS0
```

可以在KGDB内核引导启动完成后建立开发机与目标机之间的调试联系。

2．2．3 通过网络接口进行调试

kgdb也支持使用以太网接口作为调试器的连接端口。在对Linux内核应用补丁包时，需应用eth.patch补丁文件。配置内核时在Kernel hacking中选择kgdb调试项，配置kgdb调试端口为以太网接口，例如：

```
[*]KGDB: kernel debugging with remote gdb
Method for KGDB communication (KGDB: On ethernet)  --->
( ) KGDB: On generic serial port (8250)
(X) KGDB: On ethernet
```

另外使用eth0网口作为调试端口时，grub.list的配置如下：

```
title 2.6.7 kgdb
root (hd0,0)
kernel /boot/vmlinuz-2.6.7-kgdb ro root=/dev/hda1 kgdbwait kgdboe=@192.168.
5.13/,@192.168. 6.13/
```

其他的过程与使用串口作为连接端口时的设置过程相同。

注意：尽管可以使用以太网口作为kgdb的调试端口，使用串口作为连接端口更加简单易行，kgdb项目组推荐使用串口作为调试端口。

2．2．4 模块的调试方法

内核可加载模块的调试具有其特殊性。由于内核模块中各段的地址是在模块加载进内核的时候才最终确定的，所以develop机的gdb无法得到各种符号地址信息。所以，使用kgdb调试模块所需要解决的一个问题是，需要通过某种方法获得可加载模块的最终加载地址信息，并把这些信息加入到gdb环境中。

I、在Linux 2.4内核中的内核模块调试方法

在Linux2.4.x内核中，可以使用insmod -m命令输出模块的加载信息，例如：

```
[root@lisl tmp]# insmod -m hello.ko >modaddr
```

查看模块加载信息文件modaddr如下：

```
.this           00000060  c88d8000  2**2
.text           00000035  c88d8060  2**2
.rodata         00000069  c88d80a0  2**5
……
.data           00000000  c88d833c  2**2
.bss            00000000  c88d833c  2**2
……
```

在这些信息中，我们关心的只有4个段的地址:.text、.rodata、.data、.bss。在development机上将以上地址信息加入到gdb中,这样就可以进行模块功能的测试了。

```
(gdb) Add-symbol-file hello.o 0xc88d8060 -s .data 0xc88d80a0 -s
.rodata 0xc88d80a0 -s .bss 0x c88d833c
```

这种方法也存在一定的不足，它不能调试模块初始化的代码，因为此时模块初始化代码已经执行过了。而如果不执行模块的加载又无法获得模块插入地址，更不可能在模块初始化之前设置断点了。对于这种调试要求可以采用以下替代方法。

在target机上用上述方法得到模块加载的地址信息，然后再用rmmod卸载模块。在development机上将得到的模块地址信息导入到gdb环境中，在内核代码的调用初始化代码之前设置断点。这样，在target机上再次插入模块时，代码将在执行模块初始化之前停下来，这样就可以使用gdb命令调试模块初始化代码了。

另外一种调试模块初始化函数的方法是：当插入内核模块时，内核模块机制将调用函数sys_init_module(kernel/modle.c)执行对内核模块的初始化，该函数将调用所插入模块的初始化函数。程序代码片断如下：

```
……      ……
        if (mod->init != NULL)
                ret = mod->init();
……      ……
```

在该语句上设置断点，也能在执行模块初始化之前停下来。

II、在Linux 2.6.x内核中的内核模块调试方法

Linux 2.6之后的内核中，由于module-init-tools工具的更改，insmod命令不再支持-m参数，只有采取其他的方法来获取模块加载到内核的地址。通过分析ELF文件格式，我们知道程序中各段的意义如下：

.text（代码段）：用来存放可执行文件的操作指令，也就是说是它是可执行程序在内存种的镜像。

.data（数据段）：数据段用来存放可执行文件中已初始化全局变量，也就是存放程序静态分配的变量和全局变量。

.bss（BSS段）：BSS段包含了程序中未初始化全局变量，在内存中 bss段全部置零。

.rodata（只读段）：该段保存着只读数据，在进程映象中构造不可写的段。

通过在模块初始化函数中放置一下代码，我们可以很容易地获得模块加载到内存中的地址。

```
……
int bss_var;
static int hello_init(void)
{
printk(KERN_ALERT "Text location .text(Code Segment):%p\n",hello_init);
static int data_var=0;
printk(KERN_ALERT "Data Location .data(Data Segment):%p\n",&data_var);
printk(KERN_ALERT "BSS Location: .bss(BSS Segment):%p\n",&bss_var);
……
}
Module_init(hello_init);
```

这里，通过在模块的初始化函数中添加一段简单的程序，使模块在加载时打印出在内核中的加载地址。.rodata段的地址可以通过执行命令readelf -e hello.ko，取得.rodata在文件中的偏移量并加上段的align值得出。

为了使读者能够更好地进行模块的调试，kgdb项目还发布了一些脚本程序能够自动探测模块的插入并自动更新gdb中模块的符号信息。这些脚本程序的工作原理与前面解释的工作过程相似，更多的信息请阅读参考资料[4]。

2．2．5 硬件断点

kgdb提供对硬件调试寄存器的支持。在kgdb中可以设置三种硬件断点：执行断点（Execution Breakpoint）、写断点（Write Breakpoint）、访问断点（Access Breakpoint）但不支持I/O访问的断点。目前，kgdb对硬件断点的支持是通过宏来实现的，最多可以设置4个硬件断点，这些宏的用法如下：

硬件宏	含义	用法
hwebrk	设置执行断点	hwebrk breakpointno address
hwwbrk	设置写断点	hwwbrk breakpointno length address
hwabrk	设置访问断点	hwabrk breakpointno length address
hwrmbrk	删除断点	hwrmbrk breakpointno
exinfo	显示断点信息	
备注：命令中参数的含义		
Breakpointno：0~3 length:0~3 address:十六进制表示的内存地址（省略 0x 前缀）		

表 3：kgdb 硬件断点宏

在有些情况下，硬件断点的使用对于内核的调试是非常方便的。有关硬件断点的定义和具体的使用说明见参考资料[4]

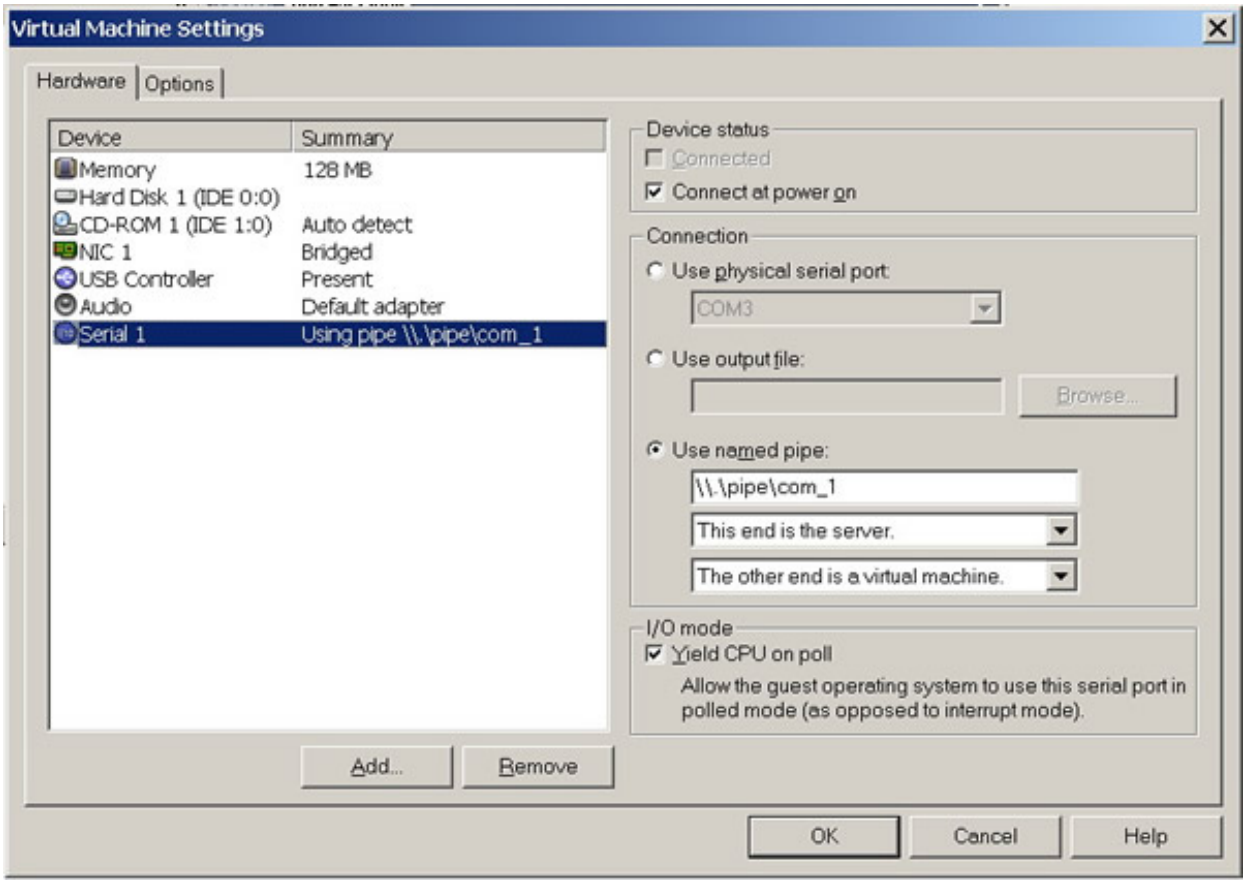
。

2．3．在VMware中搭建调试环境

kgdb调试环境需要使用两台微机分别充当development机和target机，使用VMware后我们只使用一台计算机就可以顺利完成kgdb调试环境的搭建。以windows下的环境为例，创建两台虚拟机，一台作为开发机，一台作为目标机。

2．3．1虚拟机之间的串口连接

虚拟机中的串口连接可以采用两种方法。一种是指定虚拟机的串口连接到实际的COM上，例如开发机连接到COM1，目标机连接到COM2，然后把两个串口通过串口线相连接。另一种更为简便的方法是：在较高一些版本的VMware中都支持把串口映射到命名管道，把两个虚拟机的串口映射到同一个命名管道。例如，在两个虚拟机中都选定同一个命名管道\\.\pipe\com_1,指定target机的COM口为server端，并选择"The other end is a virtual machine"属性；指定development机的COM口端为client端，同样指定COM口的"The other end is a virtual machine"属性。对于IO mode属性，在target上选中"Yield CPU on poll"复选择框，development机不选。这样，可以无需附加任何硬件，利用虚拟机就可以搭建kgdb调试环境。即降低了使用kgdb进行调试的硬件要求，也简化了建立调试环境的过程。

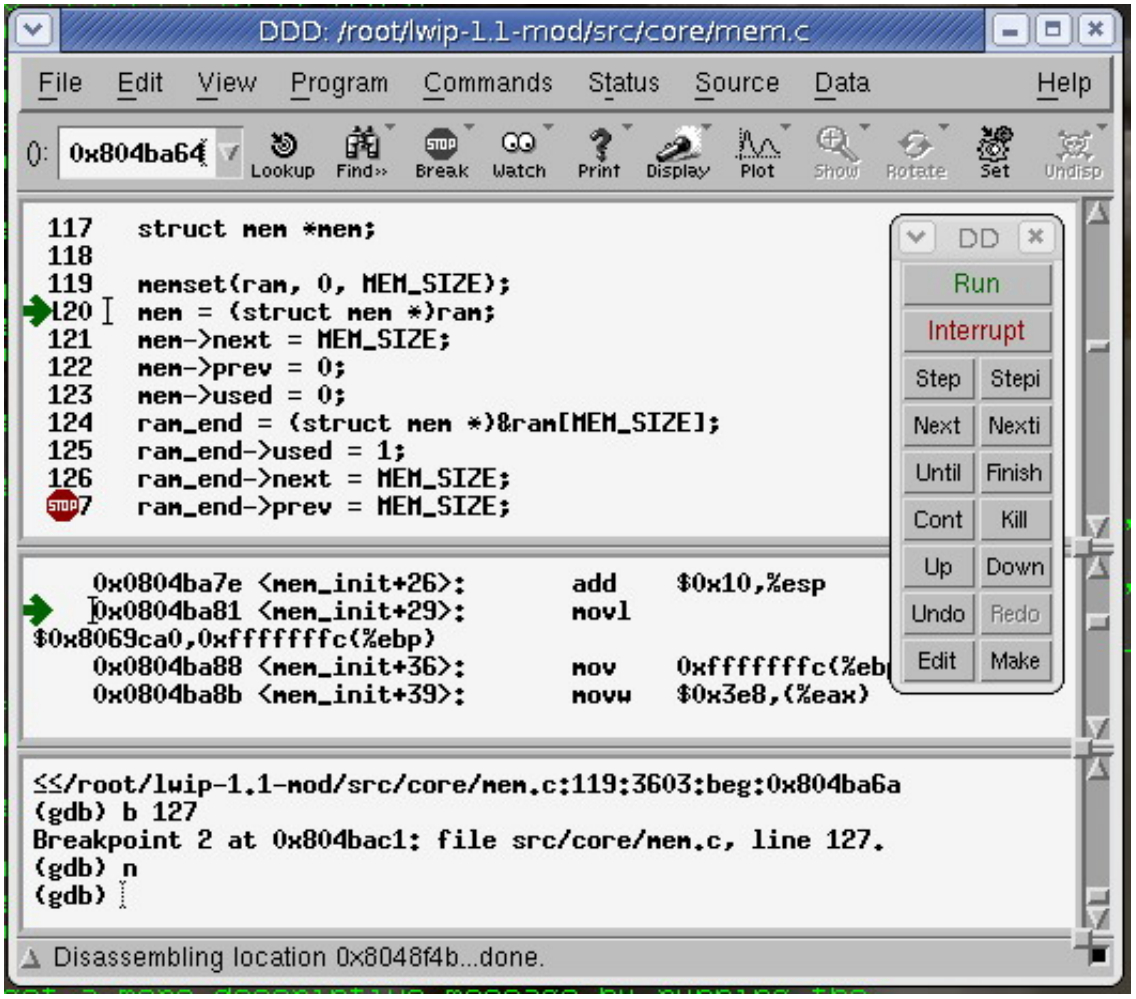


2．3．2 VMware的使用技巧

VMware虚拟机是比较占用资源的，尤其是象上面那样在Windows中使用两台虚拟机。因此，最好为系统配备512M以上的内存，每台虚拟机至少分配128M的内存。这样的硬件要求，对目前主流配置的PC而言并不是过高的要求。出于系统性能的考虑，在VMware中尽量使用字符界面进行调试工作。同时，Linux系统默认情况下开启了sshd服务，建议使用SecureCRT登陆到Linux进行操作，这样可以有较好的用户使用界面。

2．3．3 在Linux下的虚拟机中使用kgdb

对于在Linux下面使用VMware虚拟机的情况，笔者没有做过实际的探索。从原理上而言，只需要在Linux下只要创建一台虚拟机作为target机，开发机的工作可以在实际的Linux环境中进行，搭建调试环境的过程与上面所述的过程类似。由于只需要创建一台虚拟机，所以使用Linux下的虚拟机搭建kgdb调试环境对系统性能的要求较低。（vmware已经推出了Linux下的版本）还可以在development机上配合使用一些其他的调试工具，例如功能更强大的cgdb、图形界面的DDD调试器等，以方便内核的调试工作。



2．4 kgdb的一些特点和不足

使用kgdb作为内核调试环境最大的不足在于对kgdb硬件环境的要求较高，必须使用两台计算机分别作为target和development机。尽管使用虚拟机的方法可以只用一台PC即能搭建调试环境，但是对系统其他方面的性能也提出了一定的要求，同时也增加了搭建调试环境时复杂程度。另外，kgdb内核的编译、配置也比较复杂，需要一定的技巧，笔者当时做的时候也是费了很多周折。当调试过程结束后时，还需要重新制作所要发布的内核。使用kgdb并不能进行全程调试，也就是说kgdb并不能用于调试系统一开始的初始化引导过程。

不过，kgdb是一个不错的内核调试工具，使用它可以进行对内核的全面调试，甚至可以调试内核的中断处理程序。如果在一些图形化的开发工具的帮助下，对内核的调试将更方便。

---

3. 使用SkyEye构建Linux内核调试环境

SkyEye是一个开源软件项目（OPenSource Software）,SkyEye项目的目标是在通用的Linux和Windows平台上模拟常见的嵌入式计算机系统。SkyEye实现了一个指令级的硬件模拟平台，可以模拟多种嵌入式开发板，支持多种CPU指令集。SkyEye 的核心是 GNU 的 gdb 项目，它把gdb和 ARM Simulator很好地结合在了一起。加入ARMulator 的功能之后，它就可以来仿真嵌入式开发板，在它上面不仅可以调试硬件驱动，还可以调试操作系统。Skyeye项目目前已经在嵌入式系统开发领域得到了很大的推广。

3．1 SkyEye的安装和μcLinux内核编译

3．1.1 SkyEye的安装

SkyEye的安装不是本文要介绍的重点，目前已经有大量的资料对此进行了介绍。有关SkyEye的安装与使用的内容请查阅参考资料[11]。由于skyeye面目主要用于嵌入式系统领域，所以在skyeye上经常使用的是μcLinux系统，当然使用Linux作为skyeye上运行的系统也是可以的。由于介绍μcLinux 2.6在skyeye上编译的相关资料并不多，所以下面进行详细介绍。

3．1.2 μcLinux 2.6.x的编译

要在SkyEye中调试操作系统内核，首先必须使被调试内核能在SkyEye所模拟的开发板上正确运行。因此，正确编译待调试操作系统内核并配置SkyEye是进行内核调试的第一步。下面我们以SkyEye模拟基于Atmel AT91X40的开发板，并运行μcLinux 2.6为例介绍SkyEye的具体调试方法。

I、安装交叉编译环境

先安装交叉编译器。尽管在一些资料中说明使用工具链arm-elf-tools-20040427.sh ,但是由于arm-elf-xxx与arm-linux-xxx对宏及链接处理的不同，经验证明使用arm-elf-xxx工具链在链接vmlinux的最后阶段将会出错。所以这里我们使用的交叉编译工具链是：arm-uclinux-tools-base-gcc3.4.0-20040713.sh，关于该交叉编译工具链的下载地址请参见[6]。注意以下步骤最好用root用户来执行。

```
[root@lisl tmp]#chmod +x  arm-uclinux-tools-base-gcc3.4.0-20040713.sh
[root@lisl tmp]#./arm-uclinux-tools-base-gcc3.4.0-20040713.sh
```

安装交叉编译工具链之后，请确保工具链安装路径存在于系统PATH变量中。

II、制作μcLinux内核

得到μcLinux发布包的一个最容易的方法是直接访问uClinux.org站点[7]。该站点发布的内核版本可能不是最新的，但你能找到一个最新的μcLinux补丁以及找一个对应的Linux内核版本来制作一个最新的μcLinux内核。这里，将使用这种方法来制作最新的μcLinux内核。目前（笔者记录编写此文章时），所能得到的发布包的最新版本是uClinux-dist.20041215.tar.gz。

下载uClinux-dist.20041215.tar.gz，文件的下载地址请参见[7]。

下载linux-2.6.9-hsc0.patch.gz，文件的下载地址请参见[8]。

下载linux-2.6.9.tar.bz2，文件的下载地址请参见[9]。

现在我们得到了整个的linux-2.6.9源代码，以及所需的内核补丁。请准备一个有2GB空间的目录里来完成以下制作μcLinux内核的过程。

```
[root@lisl tmp]# tar -jxvf uClinux-dist-20041215.tar.bz2
[root@lisl uClinux-dist]# tar -jxvf  linux-2.6.9.tar.bz2
[root@lisl uClinux-dist]# gzip -dc linux-2.6.9-hsc0.patch.gz | patch -p0
```

或者使用：

```
[root@lisl uClinux-dist]# gunzip linux-2.6.9-hsc0.patch.gz
[root@lisl uClinux-dist]patch -p0 < linux-2.6.9-hsc0.patch
```

执行以上过程后，将在linux-2.6.9/arch目录下生成一个补丁目录 - armnommu。删除原来μcLinux目录里的linux-2.6.x(即那个linux-2.6.9-uc0)，并将我们打好补丁的Linux内核目录更名为linux-2.6.x。

```
[root@lisl uClinux-dist]# rm -rf linux-2.6.x/
[root@lisl uClinux-dist]# mv linux-2.6.9 linux-2.6.x
```

III、配置和编译μcLinux内核

因为只是出于调试μcLinux内核的目的，这里没有生成uClibc库文件及romfs.img文件。在发布μcLinux时，已经预置了某些常用嵌入式开发板的配置文件，因此这里直接使用这些配置文件，过程如下：

```
[root@lisl uClinux-dist]# cd linux-2.6.x
[root@lisl linux-2.6.x]#make ARCH=armnommu CROSS_COMPILE=arm-uclinux- atmel_
deconfig
```

atmel_deconfig文件是μcLinux发布时提供的一个配置文件，存放于目录linux-2.6.x /arch/armnommu/configs/中。

```
[root@lisl linux-2.6.x]#make ARCH=armnommu CROSS_COMPILE=arm-uclinux-
oldconfig
```

下面编译配置好的内核：

```
[root@lisl linux-2.6.x]# make ARCH=armnommu CROSS_COMPILE=arm-uclinux- v=1
```

一般情况下，编译将顺利结束并在Linux-2.6.x/目录下生成未经压缩的μcLinux内核文件vmlinux。需要注意的是为了调试μcLinux内核，需要打开内核编译的调试选项-g，使编译后的内核带有调试信息。打开编译选项的方法可以选择：

"Kernel debugging->Compile the kernel with debug info"后将自动打开调试选项。也可以直接修改linux-2.6.x目录下的Makefile文件，为其打开调试开关。方法如下：。

```
CFLAGS  += -g
```

最容易出现的问题是找不到arm-uclinux-gcc命令的错误，主要原因是PATH变量中没有包含arm-uclinux-gcc命令所在目录。在arm-linux-gcc的缺省安装情况下，它的安装目录是/root/bin/arm-linux-tool/，使用以下命令将路径加到PATH环境变量中。

```
Export PATH＝$PATH:/root/bin/arm-linux-tool/bin
```

IV、根文件系统的制作

Linux内核在启动的时的最后操作之一是加载根文件系统。根文件系统中存放了嵌入式系统使用的所有应用程序、库文件及其他一些需要用到的服务。出于文章篇幅的考虑，这里不打算介绍根文件系统的制作方法，读者可以查阅一些其他的相关资料。值得注意的是，由配置文件skyeye.conf指定了装载到内核中的根文件系统。

3．2 使用SkyEye调试

编译完μcLinux内核后，就可以在SkyEye中调试该ELF执行文件格式的内核了。前面已经说过利用SkyEye调试内核与使用gdb调试运用程序的方法相同。

需要提醒读者的是，SkyEye的配置文件 - skyeye.conf记录了模拟的硬件配置和模拟执行行为。该配置文件是SkyEye系统中一个及其重要的文件，很多错误和异常情况的发生都和该文件有关。在安装配置SkyEye出错时，请首先检查该配置文件然后再进行其他的工作。此时，所有的准备工作已经完成，就可以进行内核的调试工作了。

3．3使用SkyEye调试内核的特点和不足

在SkyEye中可以进行对Linux系统内核的全程调试。由于SkyEye目前主要支持基于ARM内核的CPU，因此一般而言需要使用交叉编译工具编译待调试的Linux系统内核。另外，制作SkyEye中使用的内核编译、配置过程比较复杂、繁琐。不过，当调试过程结束后无需重新制作所要发布的内核。

SkyEye只是对系统硬件进行了一定程度上的模拟，所以在SkyEye与真实硬件环境相比较而言还是有一定的差距，这对一些与硬件紧密相关的调试可能会有一定的影响，例如驱动程序的调试。不过对于大部分软件的调试，SkyEye已经提供了精度足够的模拟了。

SkyEye的下一个目标是和eclipse结合，有了图形界面，能为调试和查看源码提供一些方便。

---

4. 使用UML调试Linux内核

User-mode Linux（UML）简单说来就是在Linux内运行的Linux。该项目是使Linux内核成为一个运行在 Linux 系统之上单独的、用户空间的进程。UML并不是运行在某种新的硬件体系结构之上，而是运行在基于 Linux 系统调用接口所实现的虚拟机。正是由于UML是一个将Linux作为用户空间进程运行的特性，可以使用UML来进行操作系统内核的调试。有关UML的介绍请查阅参考资料[10]、[12]。

4．1 UML的安装与调试

UML的安装需要一台运行Linux 2.2.15以上，或者2.3.22以上的I386机器。对于2.6.8及其以前版本的UML，采用两种形式发布：一种是以RPM包的形式发布，一种是以源代码的形式提供UML的安装。按照UML的说明，以RPM形式提供的安装包比较陈旧且会有许多问题。以二进制形式发布的UML包并不包含所需要的调试信息，这些代码在发布时已经做了程度不同的优化。所以，要想利用UML调试Linux系统内核，需要使用最新的UML patch代码和对应版本的Linux内核编译、安装UML。完成UML的补丁之后，会在arch目录下产生一个um目录，主要的UML代码都放在该目录下。

从2.6.9版本之后（包含2.6.9版本的Linux），User-Mode Linux已经随Linux内核源代码树一起发布，它存放于arch/um目录下。

编译好UML的内核之后，直接使用gdb运行已经编译好的内核即可进行调试。

4．2使用UML调试系统内核的特点和不足

目前，用户模式 Linux 虚拟机也存在一定的局限性。由于UML虚拟机是基于Linux系统调用接口的方式实现的虚拟机，所以用户模式内核不能访问主机系统上的硬件设备。因此，UML并不适合于调试那些处理实际硬件的驱动程序。不过，如果所编写的内核程序不是硬件驱动，例如Linux文件系统、协议栈等情况，使用UML作为调试工具还是一个不错的选择。

---

5. 内核调试配置选项

为了方便调试和测试代码，内核提供了许多与内核调试相关的配置选项。这些选项大部分都在内核配置编辑器的内核开发(kernel hacking)菜单项中。在内核配置目录树菜单的其他地方也还有一些可配置的调试选项，下面将对他们作一定的介绍。

Page alloc debugging ：CONFIG_DEBUG_PAGEALLOC:

不使用该选项时，释放的内存页将从内核地址空间中移出。使用该选项后，内核推迟移出内存页的过程，因此能够发现内存泄漏的错误。

Debug memory allocations ：CONFIG_DEBUG_SLAB:

该打开该选项时，在内核执行内存分配之前将执行多种类型检查，通过这些类型检查可以发现诸如内核过量分配或者未初始化等错误。内核将会在每次分配内存前后时设置一些警戒值，如果这些值发生了变化那么内核就会知道内存已经被操作过并给出明确的提示，从而使各种隐晦的错误变得容易被跟踪。

Spinlock debugging ：CONFIG_DEBUG_SPINLOCK:

打开此选项时，内核将能够发现spinlock未初始化及各种其他的错误,能用于排除一些死锁引起的错误。

Sleep-inside-spinlock checking：CONFIG_DEBUG_SPINLOCK_SLEEP:

打开该选项时，当spinlock的持有者要睡眠时会执行相应的检查。实际上即使调用者目前没有睡眠，而只是存在睡眠的可能性时也会给出提示。

Compile the kernel with debug info ：CONFIG_DEBUG_INFO:

打开该选项时，编译出的内核将会包含全部的调试信息，使用gdb时需要这些调试信息。

Stack utilization instrumentation ：CONFIG_DEBUG_STACK_USAGE:

该选项用于跟踪内核栈的溢出错误，一个内核栈溢出错误的明显的现象是产生oops错误却没有列出系统的调用栈信息。该选项将使内核进行栈溢出检查，并使内核进行栈使用的统计。

Driver Core verbose debug messages：CONFIG_DEBUG_DRIVER:

该选项位于"Device drivers-> Generic Driver Options"下，打开该选项使得内核驱动核心产生大量的调试信息，并将他们记录到系统日志中。

Verbose SCSI error reporting (kernel size +=12K) ：CONFIG_SCSI_CONSTANTS:

该选项位于"Device drivers/SCSI device support"下。当SCSI设备出错时内核将给出详细的出错信息。

Event debugging：CONFIG_INPUT_EVBUG:

打开该选项时，会将输入子系统的错误及所有事件都输出到系统日志中。该选项在产生了详细的输入报告的同时，也会导致一定的安全问题。

以上内核编译选项需要读者根据自己所进行的内核编程的实际情况，灵活选取。在使用以上介绍的三种源代码级的内核调试工具时，一般需要选取CONFIG_DEBUG_INFO选项，以使编译的内核包含调试信息。

---

6. 总结

上面介绍了一些调试Linux内核的方法，特别是详细介绍了三种源代码级的内核调试工具，以及搭建这些内核调试环境的方法，读者可以根据自己的情况从中作出选择。

调试工具（例如gdb）的运行都需要操作系统的支持，而此时内核由于一些错误的代码而不能正确执行对系统的管理功能，所以对内核的调试必须采取一些特殊的方法进行。以上介绍的三种源代码级的调试方法，可以归纳为以下两种策略：

I、为内核增加调试Stub，利用调试Stub进行远程调试，这种调试策略需要target及development机器才能完成调试任务。

II、将虚拟机技术与调试工具相结合，使Linux内核在虚拟机中运行从而利用调试器对内核进行调试。这种策略需要制作适合在虚拟机中运行的系统内核。

由不同的调试策略决定了进行调试时不同的工作原理，同时也形成了各种调试方法不同的软硬件需求和各自的特点。

另外，需要说明的是内核调试能力的掌握很大程度上取决于经验和对整个操作系统的深入理解。对系统内核的全面深入的理解，将能在很大程度上加快对Linux系统内核的开发和调试。

对系统内核的调试技术和方法绝不止上面介绍所涉及的内容，这里只是介绍了一些经常看到和听到方法。在Linux内核向前发展的同时，内核的调试技术也在不断的进步。希望以上介绍的一些方法能对读者开发和学习Linux有所帮助。