应用调试之使用gdb和gdbserver

一、调试原理

这里的gdb调试是在pc机上对在开发板上运行的程序进行调试。具体来说的话，在pc机上要运行gdb，在开发板上运行dbserver。gdb调试的时 候，pc机上的gdb向开发板上的gdbserver发出命令，而开发板上的gdbserver就会向应用程序发出信号，使应用程序停下来或者完成其他一 些工作！由此我们知道，pc机上要运行gdb，开发板上要运行gdbserver！

二、安装gdb与gdbserver：需要 gdb-7.4.tar.bz2

gdb：

1、下载： http://ftp.gnu.org/gnu/gdb/

2、解压：tar xvf gdb-7.4.tar.bz2

3、配置：cd gdb-7.4/

./configure --target=arm-linux

4、编译：make

5、安装：mkdir tmp

make install prefix=$PWD/tmp

这里是安装到了我们当前目录的tmp里面

6、查看PC机上以前安装好的gdb版本：arm-linux-gdb -v

发现是7.4版本的，我们编译的正好也是7.4版本的哦！

7、拷贝：cp tmp/bin/arm-linux-gdb /bin/

以后我们如果想使用我们自己编译的gdb的话可以使用绝对路径：/bin/arm-linux-gdb

gdbserver

1、cd gdb/gdbserver/

2、配置： ./configure --target=arm-linux --host=arm-linux

3、编译： make CC=/usr/local/arm/3.4.5/bin/arm-linux-gcc

出现错误：

linux-arm-low.c: In function `arm_stopped_by_watchpoint':

linux-arm-low.c:642: error: `PTRACE_GETSIGINFO' undeclared (first use in this function)

linux-arm-low.c:642: error: (Each undeclared identifier is reported only once

linux-arm-low.c:642: error: for each function it appears in.)

解决方法：这里提示没有PTRACE_GETSIGINFO这个东西，这里搜索PTRACE_GETSIGINFO的路径为-I指定的头文件以及交叉 编译工具链，我们不妨到交叉编译工具链里面去查找一下：

cd /usr/local/arm/3.4.5/

grep "PTRACE_GETSIGINFO" * -nR

找到如下信息：

arm-linux/sys-include/linux/ptrace.h:27:#define PTRACE_GETSIGINFO 0x4202

arm-linux/include/linux/ptrace.h:27:#define PTRACE_GETSIGINFO 0x4202

distributed/arm-linux/sys-include/linux/ptrace.h:27:#define PTRACE_GETSIGINFO 0x4202

distributed/arm-linux/include/linux/ptrace.h:27:#define PTRACE_GETSIGINFO 0x4202

说明PTRACE_GETSIGINFO是在交叉编译工具链：linux/ptrace.h文件里定义的，那么可能是头文件没有包含好吧！

我 们到gdbserver下的linux-arm-low.c里面一看，可不是嘛，只有：#include <sys/ptrace.h>而没有：#include <linux/ptrace.h>，于是加上：#include <linux/ptrace.h>，再次编译：make CC=/usr/local/arm/3.4.5/bin/arm-linux-gcc，成功！

5. ubuntu11.10编译gdbserver时出现了linux-x86-low.c **error: sys/reg.h: No such file or directory**情况，解决方法如下：
6. 找到config.h里的HAVE_SYS_REG_H这个宏定义，把它注释掉．
7. 在linux-x86-low.c的头文件#include<sys/reg.h>注释掉．
8. 交叉编译器要指明具体路径

重新make之前先make clean一下．

4、拷贝：将gdbserver拷贝到开发板的bin目录下

三、调试

1、编译要调试的应用程序：必须要加-g选项

测试程序如下（名字是：test_debug.c）：

```
#include <stdio.h>
void C(int *p)
```

```
{
    *p = 0x12;
}

void B(int *p)
{
    C(p);
}

void A(int *p)
{
    B(p);
}

void A2(int *p)
{
    C(p);
}

int main(int argc, char **argv)
{
    int a;
    int *p = NULL;

    A2(&a);  // A2 > C
    printf("a = 0x%x\n", a);

    A(p);    // A > B > C
    return 0;
}
```

按如下编译它：arm-linux-gcc -g -o test_debug test_debug.c

2、运行时出现错误：

/mnt/code/28th_app_debug # ./test_debug

a = 0x12
Segmentation fault
下面就开始进行调试

3、在开发板上：gdbserver 192.168.183.127:2345 ./test_debug

打印出如下信息：
Process ./test_debug created; pid = 751
Listening on port 2345

其中192.168.183.127：本开发板的ip
123：端口号，自己随便写的
./test_debug：要调试的程序

4、在PC上：/bin/arm-linux-gdb ./test-debug
target remote 192.168.183.127:2345

5、下面就可以正式调试了！我们先来说一下几个常用的命令
（1）l：列出所有源代码
（2）break main：在main处打断点
break test_debug.c:11：在test_debug.c的11行打断点
（3）c：运行到断点处
（4）step：单步执行
（5）next：单步执行，但是step会进入函数里面，但是next不会
（6）print a：打印a这个变量的值
（6）quit：退出，输入此命令则开发板上的gdbserver也退出
更详细的命令，我们在下一节里面会进一步来讲讲的！

6、另一种调试方法
让程序在开发板上直接运行，当它发生错误时，令它产生core dump文件
然后使用gdb根据core dump文件找到发生错误的地方

在ARM板上：
1. ulimit -c unlimited
2. 执行应用程序：程序出错时会在当前目录下生成名为core的文件

在PC上：

3、首先将core文件拷贝到pc机上

然后：/bin/arm-linux-gdb ./test_debug ./core

打印出如下信息：

```
GNU gdb (GDB) 7.4
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/share/jz2440/test_debug...done.
[New LWP 748]
warning: `/lib/libc.so.6': Shared library architecture unknown is not compatible with
target architecture arm.
warning: `/lib/ld-linux.so.2': Shared library architecture unknown is not compatible
with target architecture arm.
Core was generated by `./test_debug'.
Program terminated with signal 11, Segmentation fault.
#0  0x000084ac in C (p=0x0) at test_debug.c:6
6               *p = 0x12;
```

4、bt：可以显示调用关系

```
#0  0x000084ac in C (p=0x0) at test_debug.c:6
#1  0x000084d0 in B (p=0x0) at test_debug.c:12
#2  0x000084f0 in A (p=0x0) at test_debug.c:17
#3  0x00008554 in main (argc=1, argv=0xbeb32eb4) at test_debug.c:34
```

gdb+gdbserver方式进行ARM程序调试

gdb+gdbserver总体介绍

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标准程串行协议协同工作，实现对目标机上的系统内核和上层应用的监控和调试功能。调试stub是嵌入式系统中的一段代码，作为宿主机GDB和目标机调试程序间的一个媒介而存在。

就目前而言，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调试目标机程序、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 的存在形式的不同，而其设计思路和实现方法则是大致相同的。

而我们最常用的是调试应用程序。就是采用gdb+gdbserver的方式进行调试。在很多情况下，用户需要对一个应用程序进行反复调试，特别是复杂的程序。采用GDB方法调试，由于嵌入式系统资源有限性，一般不能直接在目标系统上进行调试，通常采用gdb+gdbserver的方式进行调试。

一、配置编译及安装下载

1. 到http://www.gnu.org/software/gdb下载gdb-6.8.tar.gz到/tmp目录
2. 解压到/opt目录下

```
#cd /opt
#tar xzvf /tmp/gdb-6.8.tar.gz
```

3. 建立配置文件、编译

gdb允许把编译配置和编译结果放到任意的目录，因此可以在gdb目录之外建立一个专门存放编译结果目录.

```
#cd /opt
#mkdir -p arm-gdb/build
#cd arm-gdb/build
#/opt/gdb-6.8/configure --target=arm-linux --prefix=/opt/arm-gdb
#make
#make install
```

（--target配置gdb的目标平台，--prefixp指定了编译结果的存放位置，也就是安装目录。编译完后可以在/opt/arm-gdb/bin目录下找到可执行的arm-linux -gdb, arm-linux -gdbtui, arm-linux-run。 拷贝arm-linux-gdb 到/usr/bin目录

#cd /opt/arm-gdb/bin/

#cp arm-linux-gdb /usr/bin/

4. gdbserver的移植

gdbserver要用到gdb源码目录下的一些头文件，因此无法在gdb源码目录之外编译文件。

进入gdb/gdbserver目录：

[root@dding gdbserver]# pwd

/opt/gdb-6.8/gdb/gdbserver

[root@dding gdbserver]# 必须在gdbserver目录下运行配置命令，此时才能用相对路径

#./configure --target=arm-linux --host=arm-linux

（--target=arm-linux表示目标平台，--host表示主机端运行的是arm-linux-gdb，不需要配置—prefix，因为gdbserver不在主机端安装运行）

#make CC=/usr/bin/arm/4.3.2/bin/arm-linux-gcc

(这一步要指定你自己的arm-linux-gcc的绝对位置，我试过相对的不行，提示make: arm-linux-gcc: Command not found，可好多人都用的相对路径，即直接赋值arm-linux-gcc，可采取make时传递参数，也可以直接修改gdbserver目录下的Makefile文件中的环境变量CC)

没有错误的话就在gdbserver目录下生成gdbserver可执行文件，注意此时要更改其属性，否则可能会出现无法访问的情况，chmod 777 gdbserver将其更改为任何人都可以读写执行；使用arm-linux-strip命令处理一下gdbserver，将多余的符号信息删除，可让elf文件更精简，通常在应用程序的最后发布时使用；然后把它烧写到flash的根文件系统分区的/usr/bin（在此目录下，系统可以自动找到应用程序，否则必须到gdbserver所在目录下运行之），或通过nfs mount的方式都可以。只要保证gdbserver能在开发板上运行就行。

二、gdb+gdbserver nfs调试流程

下面就可以用gdb+gdbserver调试我们开发板上的程序了。在目标板上运行gdbserver，其实就是在宿主机的minicom下。连接主机和开发板.我选择了串口线和网线连接起了主机和开发板连接好电源，串口线，网线，打开串口终端.
通过NFS启动系统后,在开发板终端输入
mount -t nfs -oonolock192.168.50.72:/opt/FriendlyARM/mini2440/root_qtopia /mnt/
hello程序放在root_qtopia 下面.hello为要调试的程序（必须 - g加入调试信息）。

```
# arm-linux-gcc -g -o hello hello.c
```

要进行gdb调试，首先要在目标系统上启动gdbserver服务。在gdbserver所在目录下输入命令：

```
#cd /mnt/
#gdbserver 192.168.50.72:2345 hello
```

192.168.50.72为宿主机IP，在目标系统的2345端口（你也可以设其他可用的值，当然必须跟主机的gdb一致）开启了一个调试进程，hello为要调试的程序（必须 - g加入调试信息）。
出现提示：

```
Process /mnt/hello created: pid=80
Listening on port 2345
```

(另一个终端下)

```
#cd /opt/FriendlyARM/mini2440/root_qtopia
#arm-linux-gdb hello
```

最后一行显示：This GDB was configured as “ - - host = i686 - pc - linux - gnu，- - target = arm - linux”...，如果不一致说明arm-linux-gdb有问题
说明此gdb在X86的Host上运行，但是调试目标是ARM代码。

(gdb) target remote 192.168.50.168:2345

（192.168.50.168为开发板IP）

Remote debugging using 192.168.50.168:2345

[New thread 80]

[Switching to thread 80]

0x40002a90 in ??()

同时在minicom下提示：

Remote debugging from host 192.168.50.72(gdb)

注意：你的端口号必须与gdbserver开启的端口号一致，这样才能进行通信。建立链接后，就可以进行调试了。调试在Host端，跟gdb调试方法相同。注意的是要用“c”来执行命令，不能用“r”。因为程序已经在Target Board上面由gdbserver启动了。结果输出是在Target Board端，用超级终端查看。连接成功，这时候就可以输入各种GDB命令如list、run、next、step、break等进行程序调试了。

建立连接后进行gdb远程调试和gdb本地调试方法相同