下载gdb:

<https://ftp.gnu.org/gnu/gdb/>

现在最新的是10.1.

我们下载视频中说的：

gdb-7.4.tar.gz

然后解压：tar xjf gdb-7.4.tar.gz

我们的gdb在pc 我们要调试的代码在arm 板上

那莫我们如何发调试信息给app 呢？

app 有个父进程叫做gdbserver.它位于arm板上。它是被调试程序的父进程

gdb 会发消息给gdb server。

假如我们要查看app 某个变量a的地址。我们需要知道a变量的地址，所以我们需要一个带有调试信息的app.

进入gdb\_7.4 然后执行./configure –help

可以看到如下的信息：

System types:

--build=BUILD configure for building on BUILD [guessed]

--host=HOST cross-compile to build programs to run on HOST [BUILD]

--target=TARGET configure for building compilers for TARGET [HOST]

Host 我们表示编译出来的程序在那里运行

Target 表示我们的程序时用于什么。

我们配置如下：

./configure --target=arm-linux

make

如果编译有如下的错误：

opncls.c: In function 'bfd\_fopen':   
bfd.h:537:65: error: right-hand operand of comma expression has no effect [-Werror=unused-value]

我们要在配置的时候：./configure --target=arm-linux –disable-werror

直接屏蔽掉werror报警。

我们make install 的时候；

默认的安装路径：prefix = /usr/local

我们改为临时的路径tmp 在gdb 里面新建一个文件

make install prefix=$pwd/tmp 也就是当前目录下的tmp

会出现错误：

Makefile:371: recipe for target 'gdb.info' failed

make[5]: \*\*\* [gdb.info] Error 1

make[5]: Leaving directory '/home/book/s3c2440/app\_debug/gdb-7.4/gdb/doc'

Makefile:1298: recipe for target 'subdir\_do' failed

make[4]: \*\*\* [subdir\_do] Error 1

make[4]: Leaving directory '/home/book/s3c2440/app\_debug/gdb-7.4/gdb'

Makefile:1025: recipe for target 'install-only' failed

make[3]: \*\*\* [install-only] Error 2

make[3]: Leaving directory '/home/book/s3c2440/app\_debug/gdb-7.4/gdb'

Makefile:1022: recipe for target 'install' failed

make[2]: \*\*\* [install] Error 2

make[2]: Leaving directory '/home/book/s3c2440/app\_debug/gdb-7.4/gdb'

Makefile:8428: recipe for target 'install-gdb' failed

make[1]: \*\*\* [install-gdb] Error 2

make[1]: Leaving directory '/home/book/s3c2440/app\_debug/gdb-7.4'

Makefile:2079: recipe for target 'install' failed

make: \*\*\* [install] Error 2

可以用arm-linux-gdb –v查看是否我们有gdb 发现我们没有

进入到我们新创建的tmp 目录下 bin

可以查看我们的gdb 版本：7.4

然后进行如下的操作：

sudo cp arm-linux-gdb /bin/

sudo chmod +x /bin/arm-linux-gdb

编译gdb server:

进入gdb/gdbserver:

配置：

./configure –host=arm-linux //没发生错误

make

找不到一个PTRACE\_GETSTGINFO

头文件找不到除了在-I 指定的路径下 还在交叉编译链中找。

我们通过echo $PATH //查看交叉编译链的库

我们自己的交叉编译链：

/work/tools/gcc-3.4.5-glibc-2.3.6/

进入交叉编译链：

Cd /work/tools/gcc-3.4.5-glibc-2.3.6 //因为我们呢

查找这个宏 grep “xxx” \* -nR

在linux/patch.h有定义

打开 vi linux-arm-low.c 查看头文件

发现确实没有包含头文件linux/patch.h

将#include <sys/ptrace.h> 改为linux/patch.h //不要用这种方式

我们新加一行linux/patch.h

将生成的gdbserver 拷贝到我们的网络文件系统中的的/bin

编译要调试的应用程序要加上-g 选项。否则的话gdb server 不知道gdb 要做的事情是什么。

我们在写应用程序故意引入错误。

编译加上-g

arm-linux-gcc -g -o test\_debug test\_debug.c

拷贝到开发板：

在开发板执行启动gdbserver 并运行应用程序：

gdbserver 192.168.0.11:2345 ./test\_debug

在pc 上：

/bin/arm-linux-gdb ./test\_debug //我们自己安装的gdb 在/bin下

然后跟开发板建立连接：

Target remote 192.168.0.11:2345

l 可以查看源码

break main main打断点

c //continue 意思

可以在任意位置打断点：break test\_debug：31

step 单步执行

然后 通过print 打印出变量

print \*p

这样就可以调试。

2.

假如我们不想通过这种方式。让程序直接运行，当他运行错误式，产生core dump

然后用gdb 根据core dump 文件进行调试。

可以查下core dump 的使用。

ulimit –c

可以发现是0 也就是不产生core dump 文件

改为不限制：

Ulimit –c unlimited

执行应用程序。这样就产生了core demp

拷贝到我们pc机：

cp: cannot open '/work/nfs\_root/first\_fs/core' for reading: Permission denied

显示权限不足的时候我们一定要在开发板中执行命令：

加上权限 chmod 777 core

在pc上执行.后启动gdb

/bin/arm-linux-gdb ./test\_debug ./core

看到如下的错误：

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;

利用bt 命令可以查看调用关系

#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=0xbeea1ee4) at test\_debug.c:34