根据内核的打印出来的oops信息调试：

我们在驱动程序中不能直接使用物理地址。假如我们故意的引入错误。

不用ioremap直接给一个地址。我们加载这个驱动然后用./firsttest on 去测试的话就会显示段错误。

如下：

/\*无法处理页请求也就是无法处理这个虚拟地址56000050\*/

Unable to handle kernel paging request at virtual address 56000050

pgd = c3d34000

[56000050] \*pgd=00000000

Internal error: Oops: 5 [#1]

/\*哪个模块出现问题\*/

Modules linked in: first\_drv

CPU: 0 Not tainted (2.6.22.6 #1)

/\*pc就是发生错误的时候的指令地址\*/

PC is at first\_drv\_open+0x18/0x3c [first\_drv]

/\*LR寄存器\*/

LR is at chrdev\_open+0x14c/0x164

/\*执行这个错误的各个寄存器的值\*/

pc : [<bf000018>] lr : [<c008d888>] psr: a0000013

sp : c3ec5e88 ip : c3ec5e98 fp : c3ec5e94

r10: 00000000 r9 : c3ec4000 r8 : c3ec63e0

r7 : 00000000 r6 : 00000000 r5 : c3e9b0c0 r4 : c06f0120

r3 : bf000000 r2 : 56000050 r1 : bf000964 r0 : 00000000

Flags: NzCv IRQs on FIQs on Mode SVC\_32 Segment user

Control: c000717f Table: 33d34000 DAC: 00000015

/\*发生错误的时候当前进程的名称\*/

Process firsttest (pid: 775, stack limit = 0xc3ec4258)

/\*栈可以从栈中分析出函数的调用关系\*/

Stack: (0xc3ec5e88 to 0xc3ec6000)

5e80: c3ec5ebc c3ec5e98 c008d888 bf000010 00000000 c3ec63e0

5ea0: c3e9b0c0 c008d73c c0474e20 c3e9d24c c3ec5ee4 c3ec5ec0 c0089e48 c008d74c

5ec0: c3ec63e0 c3ec5f04 00000003 ffffff9c c002c044 c04de000 c3ec5efc c3ec5ee8

5ee0: c0089f64 c0089d58 00000000 00000002 c3ec5f68 c3ec5f00 c0089fb8 c0089f40

5f00: c3ec5f04 c3e9d24c c0474e20 00000000 00000000 c3d35000 00000101 00000001

5f20: 00000000 c3ec4000 c046d8c8 c046d8c0 ffffffe8 c04de000 c3ec5f68 c3ec5f48

5f40: c008a16c c009fc70 00000003 00000000 c3ec63e0 00000002 bee4aee0 c3ec5f94

5f60: c3ec5f6c c008a2f4 c0089f88 00008520 bee4aed4 0000860c 00008670 00000005

5f80: c002c044 4013365c c3ec5fa4 c3ec5f98 c008a3a8 c008a2b0 00000000 c3ec5fa8

5fa0: c002bea0 c008a394 bee4aed4 0000860c 00008720 00000002 bee4aee0 00000001

5fc0: bee4aed4 0000860c 00008670 00000002 00008520 00000000 4013365c bee4aea8

5fe0: 00000000 bee4ae84 0000266c 400c98e0 60000010 00008720 00766564 00000000

/\*回溯从下面的分析是这样的调用方式sys\_open do\_sys\_open o\_filp\_open \*/

Backtrace:

[<bf000000>] (first\_drv\_open+0x0/0x3c [first\_drv]) from [<c008d888>] (chrdev\_open+0x14c/0x164)

[<c008d73c>] (chrdev\_open+0x0/0x164) from [<c0089e48>] (\_\_dentry\_open+0x100/0x1e8)

r8:c3e9d24c r7:c0474e20 r6:c008d73c r5:c3e9b0c0 r4:c3ec63e0

[<c0089d48>] (\_\_dentry\_open+0x0/0x1e8) from [<c0089f64>] (nameidata\_to\_filp+0x34/0x48)

[<c0089f30>] (nameidata\_to\_filp+0x0/0x48) from [<c0089fb8>] (do\_filp\_open+0x40/0x48)

r4:00000002

[<c0089f78>] (do\_filp\_open+0x0/0x48) from [<c008a2f4>] (do\_sys\_open+0x54/0xe4)

r5:bee4aee0 r4:00000002

[<c008a2a0>] (do\_sys\_open+0x0/0xe4) from [<c008a3a8>] (sys\_open+0x24/0x28)

[<c008a384>] (sys\_open+0x0/0x28) from [<c002bea0>] (ret\_fast\_syscall+0x0/0x2c)

Code: e24cb004 e59f1024 e3a00000 e5912000 (e5923000)

Segmentation fault

上面的回溯信息要在内核menu config的时候配置上选项才能看到。

可以查找FRAME\_POINTER

Symbol: FRAME\_POINTER [=y]

Prompt: Compile the kernel with frame pointers

Defined at lib/Kconfig.debug:357

Depends on: DEBUG\_KERNEL && (X86 || CRIS || M68K || M68KNOMMU || FRV || UML || S390 || AVR32 || SUPE │

Location:

Kernel hacking

Kernel debugging (DEBUG\_KERNEL [=y])

Selected by: LOCKDEP && DEBUG\_KERNEL && TRACE\_IRQFLAGS\_SUPPORT && STACKTRACE\_SUPPORT && LOCKDEP\_SUPP

在Kernel hacking 下要进行配置不配置的的话我们就得通过对栈的分析从而得出函数的调用关系。

大多时候pc只是给出一个地址并不会告诉是哪个函数发生错误：

上面的pc内容如下：

PC is at first\_drv\_open+0x18/0x3c [first\_drv]

他的意思是：

在first\_drv\_open + 0x18地址发生错误 0x3c 是函数的总大小

根据pc值确定该指令属于内核还是外加的模块

pc=: 0xbf000018

它属于什么的地址是内核的还是insmod 加载的驱动程序？

先判断他是否属于内核的地址，内核地址通过system.map确定内核的函数的地址范围。

如果不属于内核system.map的地址范围则属于insmod加载的驱动程序。

1. 如果确定是insmod引入的错误，但是pc并没有告诉是哪个函数，怎末做？

cat /proc/kallsyms（内核的函数加载的函数地址）

将其中的内容从定向到cat /proc/kallsyms > /kallsyms.txt

kallsyms.txt 里面的内容：

t 静态函数 修饰

T void 函数

从这些信息中周到一个相近的地址，地址<= bf000018

从中可以找到：

bf000000 t first\_drv\_open [first\_drv] 他表示的意思是first\_drv\_open是属于first\_drv这个驱动程序，他的地址是bf000000。

由段错误的输出信息：

first\_drv\_open + 0x18地址发生错误可以直到他是在bf000000 + 18的地方发生错误。

所以就能确定是first\_drv.ko驱动发生了错误

在pc上反汇编：arm-linux-objdump –D first\_drv.ko > first\_drv.dis

因为开发板上没有linux-objdump。

在反汇编中找到first\_drv\_open如下：

Ko 文件里 insmod 后

00000000 <first\_drv\_open>: bf000000 t first\_drv\_open

00000018 pc = bf000018

反汇编的如下

00000000 <first\_drv\_open>:

0: e1a0c00d mov ip, sp

4: e92dd800 stmdb sp!, {fp, ip, lr, pc}

8: e24cb004 sub fp, ip, #4 ; 0x4

/\*假如我们在menuconfig的时候配置了FRAME\_POINTER也就是fp的值，因为sub fp, ip,将ip 放到fp ip 又代表着sp的值。所以从fp就可以得到sp的信息 \*/

c: e59f1024 ldr r1, [pc, #36] ; 38 <\_\_mod\_vermagic5>

10: e3a00000 mov r0, #0 ; 0x0

14: e5912000 ldr r2, [r1]

18: e5923000 ldr r3, [r2] /\*出错 从 r2的地址取值放到r3 r2=56000050\*/

1c: e3c33c3f bic r3, r3, #16128 ; 0x3f00

20: e5823000 str r3, [r2]

/\*这三句的意思就是\*gpfcon &= ~((0x3<<(4\*2)) | (0x3<<(5\*2)) | (0x3<<(6\*2))); \*/

24: e5912000 ldr r2, [r1]

28: e5923000 ldr r3, [r2]

2c: e3833c15 orr r3, r3, #5376 ; 0x1500

30: e5823000 str r3, [r2]

34: e89da800 ldmia sp, {fp, sp, pc}

38: 00000000 andeq r0, r0, r0

将有问题的驱动拷贝到 drivers/char下

修改driver/char/makefile

加上obj += first\_drv.o

将有问题的驱动作为模块编进内核。用新的uimage 启动。继续测试：

同样找到pc 出错误的地方。

根据pc值确定该指令属于内核还是外加的模块

Pc =c014e6c0 vi /System.map 确定是属于内核的范围

反汇编内核：arm-linux-objdump –D vmlinux > vmlinux.dis //会很大

在dis文件中搜索c014e6c0：

找到相应的出问题的函数