# lab9\_fasync实验说明

通过本实验学会如何给一个字符设备驱动程序添加异步通知功能。

## 基本实验步骤

1. 进入rlk\_lab/rlk\_basic/chapter\_5/lab9目录。

|  |
| --- |
| # export ARCH=arm  # export CROSS\_COMPILE=arm-linux-gnueabi-  # make BASEINCLUDE=/home/figo/work/runninglinuxkernel/runninglinuxkernel\_4.0 |

这里BASEINCLUDE指定你当前runninglinuxkernel\_4.0的目录路径。

编译test测试app。

|  |
| --- |
| # arm-linux-gnueabi-gcc test.c -o test |

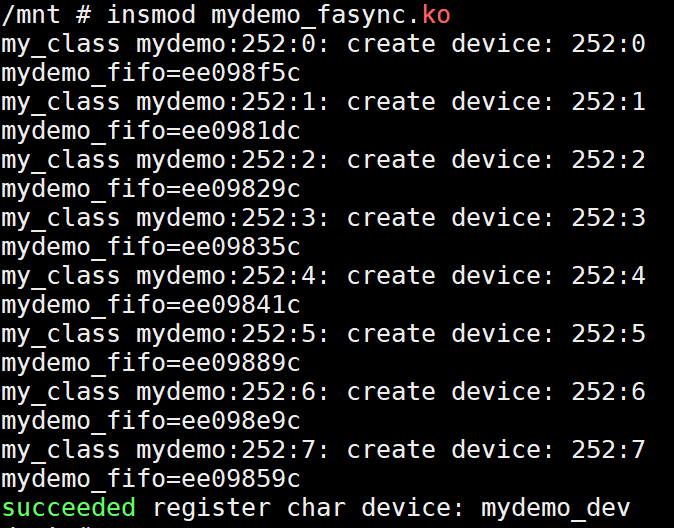
然后把mydemo\_fasync.ko和test拷贝到runninglinuxkernel\_4.0/kmodules目录下面。

运行如下脚本启动Qemu。

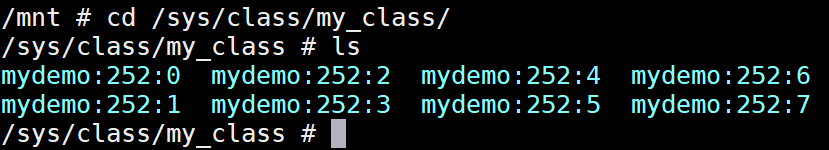
|  |
| --- |
| #cd runninglinuxkernel\_4.0  # sh run.sh arm32 #启动虚拟机 |

在Qemu虚拟机:

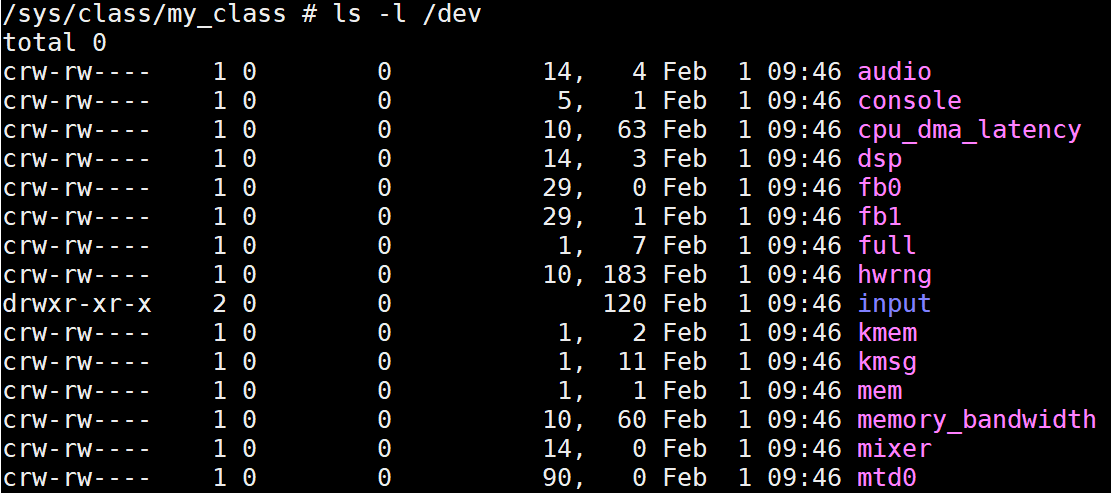
|  |
| --- |
| #cd /mnt  # insmod mydemo\_fasync.ko |



你会看到创建了8个设备。你可以到/sys/class/my\_class/目录下面看到这些设备。



我们可以看到创建了主设备号为252的设备。我们再来看一下/dev/目录。



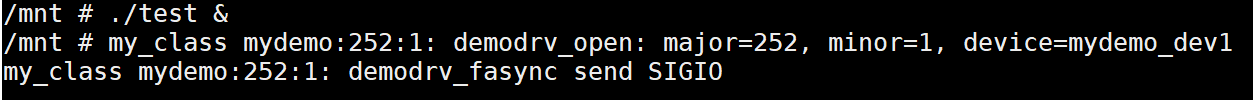
发现并没有主设备为252的设备。

所以我们需要手工创建一个设备用来test app。

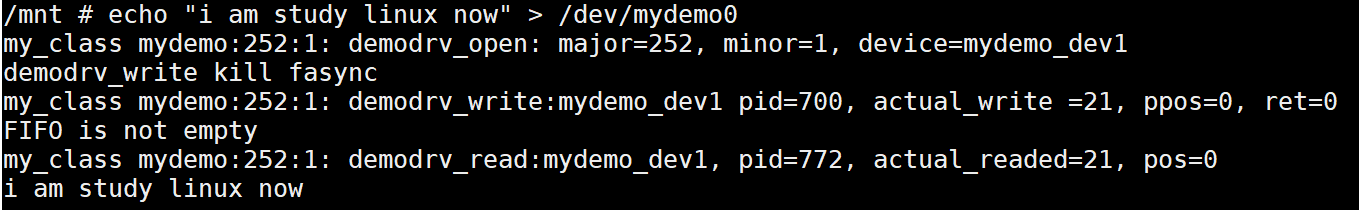
|  |
| --- |
| #mknod /dev/mydemo0 c 252 1 |

接下来跑我们的test程序：

|  |
| --- |
| # ./test & #这里让test程序在后台跑 |



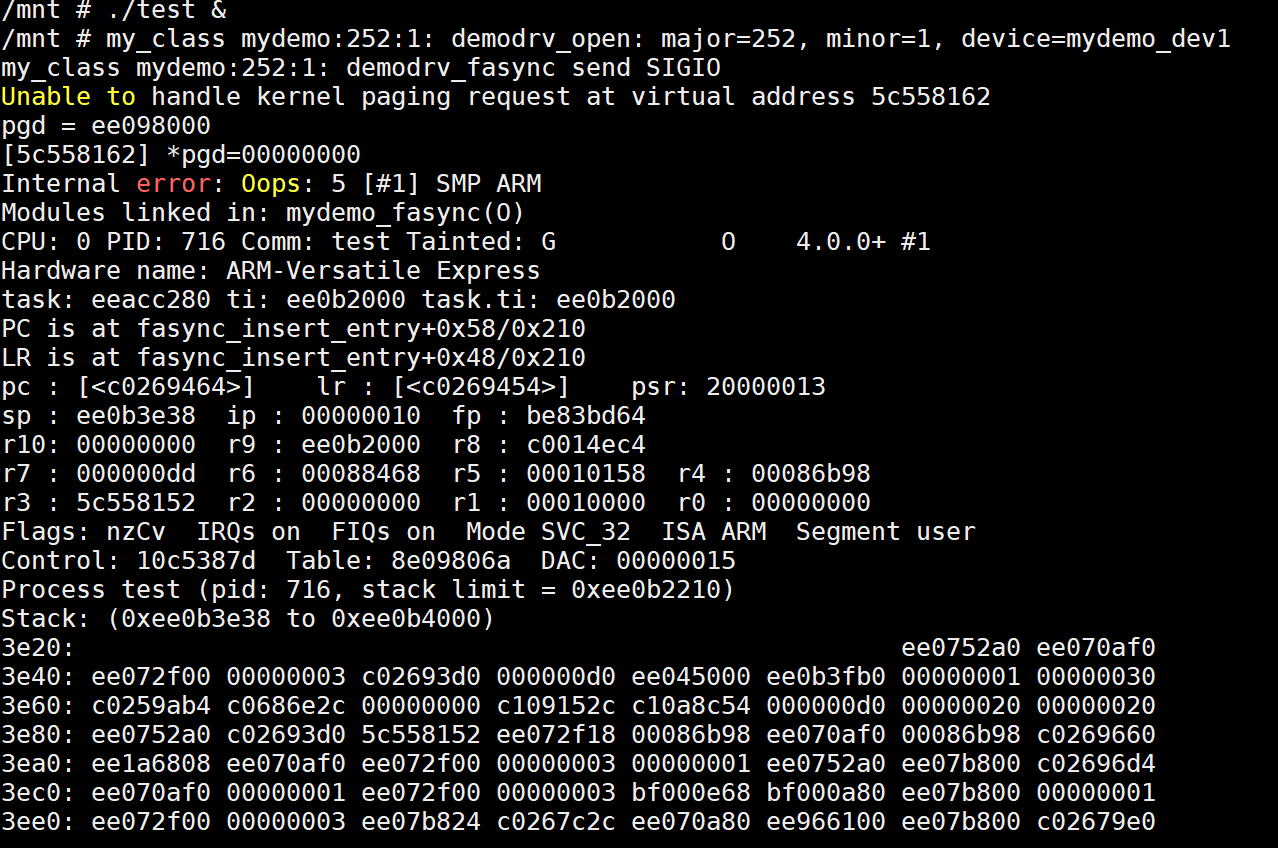
然后使用echo命令来往/dev/mydemo0这个设备写入字符串。

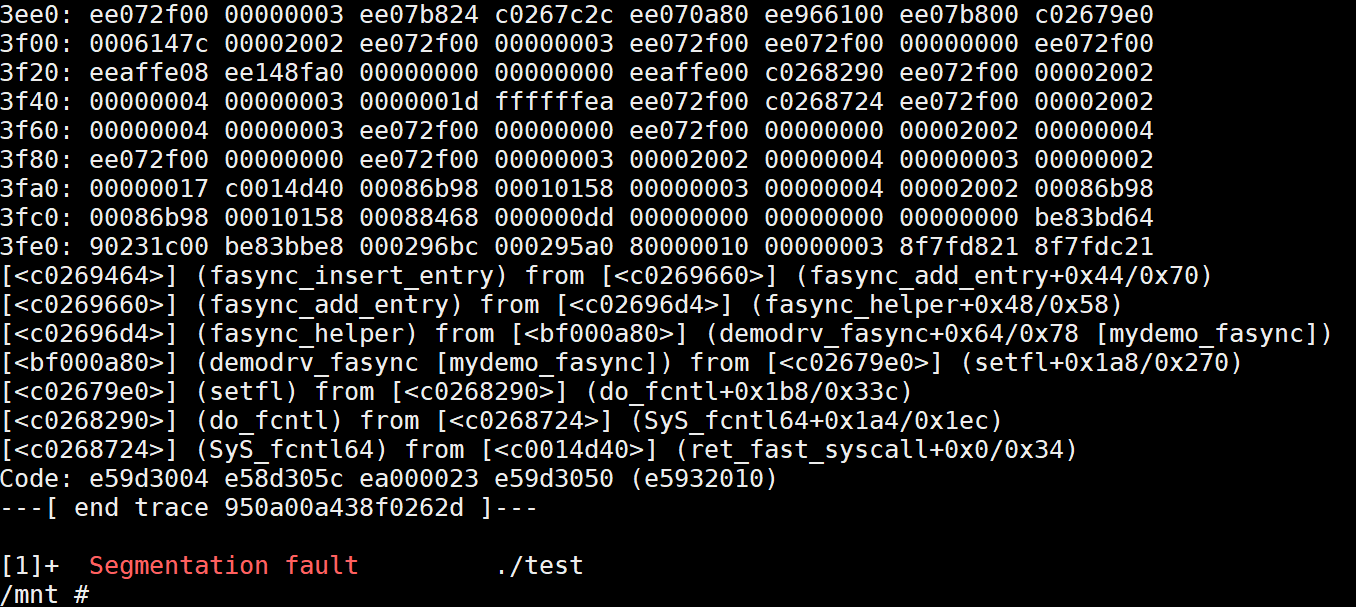


可以看到从demodrv\_read()函数把刚才写入的字符串已经读到用户空间了。

## 进阶思考

在这个实验里，小明和小李同时做这个实验，小李得到了正确的结果，而小明却没有，他运行test程序之后，发生了Oops错误。





log如下：

|  |
| --- |
| /mnt # ./test &  /mnt # my\_class mydemo:252:1: demodrv\_open: major=252, minor=1, device=mydemo\_dev1  my\_class mydemo:252:1: demodrv\_fasync send SIGIO  Unable to handle kernel paging request at virtual address 5c558162  pgd = ee098000  [5c558162] \*pgd=00000000  Internal error: Oops: 5 [#1] SMP ARM  Modules linked in: mydemo\_fasync(O)  CPU: 0 PID: 716 Comm: test Tainted: G O 4.0.0+ #1  Hardware name: ARM-Versatile Express  task: eeacc280 ti: ee0b2000 task.ti: ee0b2000  PC is at fasync\_insert\_entry+0x58/0x210  LR is at fasync\_insert\_entry+0x48/0x210  pc : [<c0269464>] lr : [<c0269454>] psr: 20000013  sp : ee0b3e38 ip : 00000010 fp : be83bd64  r10: 00000000 r9 : ee0b2000 r8 : c0014ec4  r7 : 000000dd r6 : 00088468 r5 : 00010158 r4 : 00086b98  r3 : 5c558152 r2 : 00000000 r1 : 00010000 r0 : 00000000  Flags: nzCv IRQs on FIQs on Mode SVC\_32 ISA ARM Segment user  Control: 10c5387d Table: 8e09806a DAC: 00000015  Process test (pid: 716, stack limit = 0xee0b2210)  Stack: (0xee0b3e38 to 0xee0b4000)  3e20: ee0752a0 ee070af0  3e40: ee072f00 00000003 c02693d0 000000d0 ee045000 ee0b3fb0 00000001 00000030  3e60: c0259ab4 c0686e2c 00000000 c109152c c10a8c54 000000d0 00000020 00000020  3e80: ee0752a0 c02693d0 5c558152 ee072f18 00086b98 ee070af0 00086b98 c0269660  3ea0: ee1a6808 ee070af0 ee072f00 00000003 00000001 ee0752a0 ee07b800 c02696d4  3ec0: ee070af0 00000001 ee072f00 00000003 bf000e68 bf000a80 ee07b800 00000001  3ee0: ee072f00 00000003 ee07b824 c0267c2c ee070a80 ee966100 ee07b800 c02679e0  3f00: 0006147c 00002002 ee072f00 00000003 ee072f00 ee072f00 00000000 ee072f00  3f20: eeaffe08 ee148fa0 00000000 00000000 eeaffe00 c0268290 ee072f00 00002002  3f40: 00000004 00000003 0000001d ffffffea ee072f00 c0268724 ee072f00 00002002  3f60: 00000004 00000003 ee072f00 00000000 ee072f00 00000000 00002002 00000004  3f80: ee072f00 00000000 ee072f00 00000003 00002002 00000004 00000003 00000002  3fa0: 00000017 c0014d40 00086b98 00010158 00000003 00000004 00002002 00086b98  3fc0: 00086b98 00010158 00088468 000000dd 00000000 00000000 00000000 be83bd64  3fe0: 90231c00 be83bbe8 000296bc 000295a0 80000010 00000003 8f7fd821 8f7fdc21  [<c0269464>] (fasync\_insert\_entry) from [<c0269660>] (fasync\_add\_entry+0x44/0x70)  [<c0269660>] (fasync\_add\_entry) from [<c02696d4>] (fasync\_helper+0x48/0x58)  [<c02696d4>] (fasync\_helper) from [<bf000a80>] (demodrv\_fasync+0x64/0x78 [mydemo\_fasync])  [<bf000a80>] (demodrv\_fasync [mydemo\_fasync]) from [<c02679e0>] (setfl+0x1a8/0x270)  [<c02679e0>] (setfl) from [<c0268290>] (do\_fcntl+0x1b8/0x33c)  [<c0268290>] (do\_fcntl) from [<c0268724>] (SyS\_fcntl64+0x1a4/0x1ec)  [<c0268724>] (SyS\_fcntl64) from [<c0014d40>] (ret\_fast\_syscall+0x0/0x34)  Code: e59d3004 e58d305c ea000023 e59d3050 (e5932010)  ---[ end trace 950a00a438f0262d ]---  [1]+ Segmentation fault ./test |

这是为什么呢？请您帮小明解决一下这个问题，分析这个问题产生的原因和给出解决办法。

如果大家对这个问题感兴趣，可以关注笨叔的第一季旗舰篇视频，笨叔会在视频中和大家详细解答。

请关注奔跑吧Linux社区微信公众号，里面有更多更有料更好玩的Linux干货：



奔跑吧Linux社区的旗舰篇视频节目，帮您打通Linux和职场任督二脉！





笨叔最新力作，全程约5小时高清，140多页ppt，8大实验，基于x86\_64的Centos 7.6和arm64，提供全套实验素材和环境。全面介绍kdump+crash在死机黑屏方面的实战应用，全部案例源自线上云服务器和嵌入式产品开发实际案例！

不能为客户和老板解决死机黑屏问题的都不是合格的linux运维或者研发攻城狮！





订阅视频请到淘宝店：<https://shop115683645.taobao.com>

