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
```

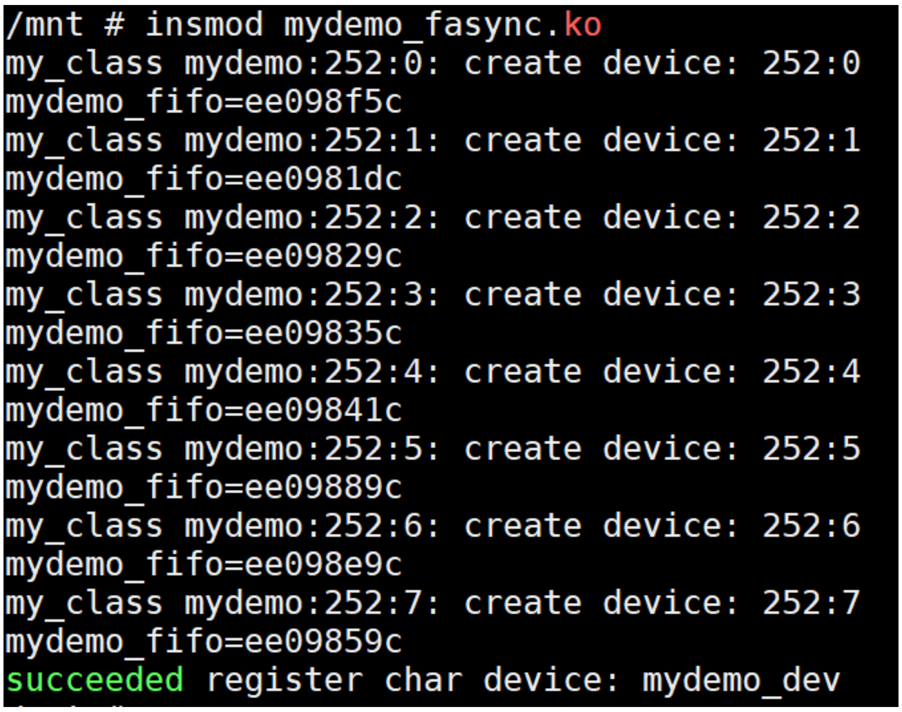
然后把 mydemo_fasync.ko 和 test 拷贝到 runninglinuxkernel_4.0/kmodules 目录下面。

运行如下脚本启动 Qemu。

```
#cd runninglinuxkernel_4.0
# sh run.sh arm32    #启动虚拟机
```

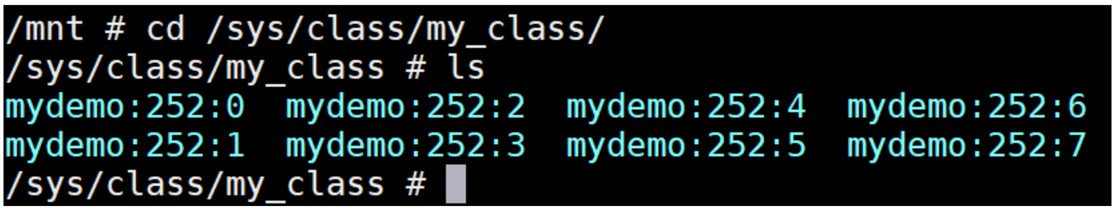
在 Qemu 虚拟机:

```
#cd /mnt
# insmod mydemo_fasync.ko
```



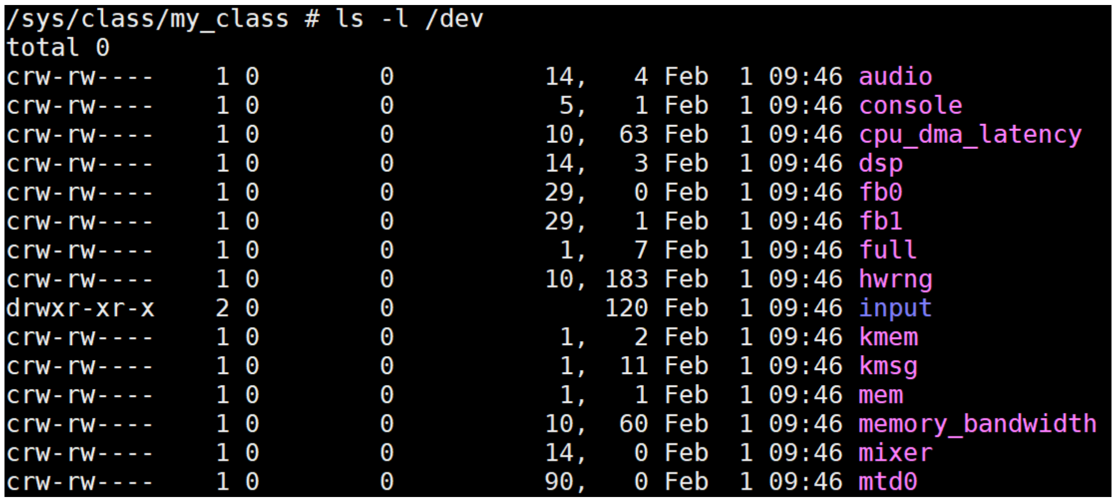
```
/mnt # insmod mydemo_fasync.ko
my_class mydemo:252:0: create device: 252:0
mydemo_fifo=ee098f5c
my_class mydemo:252:1: create device: 252:1
mydemo_fifo=ee0981dc
my_class mydemo:252:2: create device: 252:2
mydemo_fifo=ee09829c
my_class mydemo:252:3: create device: 252:3
mydemo_fifo=ee09835c
my_class mydemo:252:4: create device: 252:4
mydemo_fifo=ee09841c
my_class mydemo:252:5: create device: 252:5
mydemo_fifo=ee09889c
my_class mydemo:252:6: create device: 252:6
mydemo_fifo=ee098e9c
my_class mydemo:252:7: create device: 252:7
mydemo_fifo=ee09859c
succeeded register char device: mydemo_dev
```

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。



```
/mnt # cd /sys/class/my_class/
/sys/class/my_class # ls
mydemo:252:0  mydemo:252:2  mydemo:252:4  mydemo:252:6
mydemo:252:1  mydemo:252:3  mydemo:252:5  mydemo:252:7
/sys/class/my_class #
```

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。



```
/sys/class/my_class # ls -l /dev
total 0
crw-rw----    1 0        0          14,   4 Feb  1 09:46 audio
crw-rw----    1 0        0           5,   1 Feb  1 09:46 console
crw-rw----    1 0        0          10,  63 Feb  1 09:46 cpu_dma_latency
crw-rw----    1 0        0          14,   3 Feb  1 09:46 dsp
crw-rw----    1 0        0          29,   0 Feb  1 09:46 fb0
crw-rw----    1 0        0          29,   1 Feb  1 09:46 fb1
crw-rw----    1 0        0           1,   7 Feb  1 09:46 full
crw-rw----    1 0        0          10, 183 Feb  1 09:46 hwrng
drwxr-xr-x    2 0        0              120 Feb  1 09:46 input
crw-rw----    1 0        0           1,   2 Feb  1 09:46 kmem
crw-rw----    1 0        0           1,  11 Feb  1 09:46 kmsg
crw-rw----    1 0        0           1,   1 Feb  1 09:46 mem
crw-rw----    1 0        0          10,  60 Feb  1 09:46 memory_bandwidth
crw-rw----    1 0        0          14,   0 Feb  1 09:46 mixer
crw-rw----    1 0        0          90,   0 Feb  1 09:46 mtd0
```

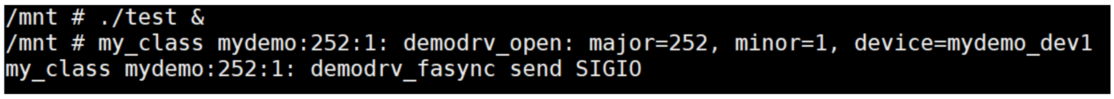
发现并没有主设备为 252 的设备。

所以我们需要手工创建一个设备用来 test app。

```
#mknod /dev/mydemo0 c 252 1
```

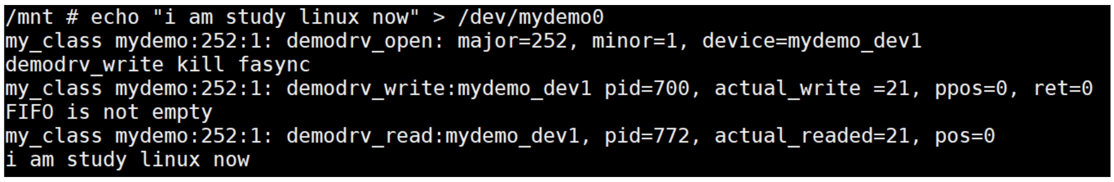
接下来跑我们的 test 程序：

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



```
/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
```

然后使用 echo 命令来往/dev/mydemo0 这个设备写入字符串。

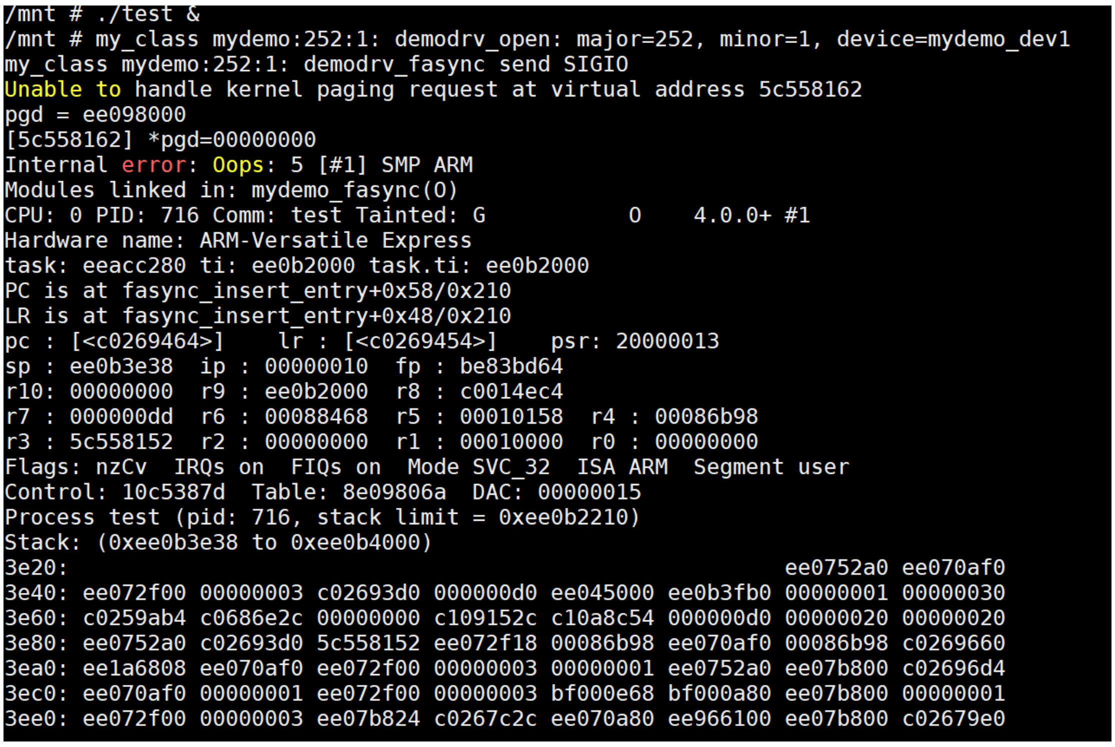


```
/mnt # echo "i am study linux now" > /dev/mydemo0
my_class mydemo:252:1: demodrv_open: major=252, minor=1, device=mydemo_dev1
demodrv_write kill fasync
my_class mydemo:252:1: demodrv_write:mydemo_dev1 pid=700, actual_write =21, ppos=0, ret=0
FIFO is not empty
my_class mydemo:252:1: demodrv_read:mydemo_dev1, pid=772, actual_readed=21, pos=0
i am study linux now
```

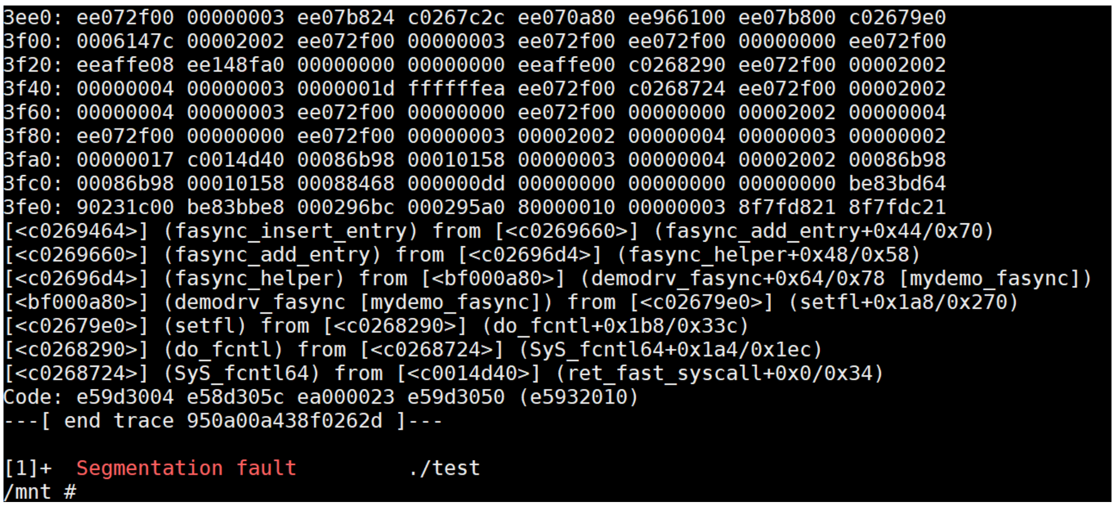
可以看到从 demodrv_read()函数把刚才写入的字符串已经读到用户空间了。

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。



```
/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
```



```
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
/mnt #
```

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。

shop115683645. taobao. com
配套视频 旗舰篇
第 1 季
内存管理
奔跑吧 Linux内核
奔跑吧 LINUX社区
旗舰篇一次订阅， 持续更新
规划中
第二季 进程管理和调度 / 中断 / 锁 （已出）
第三季 虚拟化
第四季 Linux 内核和应用开发调试必杀技
第五季 红帽系列

shop115683645.taobao.com
Linux视频课程
1. 一键订阅， 持续更新
2. 最有深度和广度的 Linux 视频
3. 手把手解读 Linux 内核代码
4. 紧跟 Linux 开源社区技术热点
5. 笨叔叔的 VIP 私密群答疑
6. 图书 + 视频， 全新学习模式
奔跑吧 LINUX社区
微信公众号： 奔跑吧 linux 社区

微店：

微店 奔跑吧Linux内核

扫码识别

淘宝店：https://shop115683645.taobao.com/

微信公众号：