# Linux块设备驱动程序实验指导书

本次实验将完成下列过程：

（1）编写块设备驱动程序并编译。

（2）安装并切换Linux内核至指定版本。

（3）测试块设备驱动程序。

本次实验基于以下软件和环境完成：

（1）VMware® Workstation 16 Pro

（2）Ubuntu 20.04 LTS

（3）Linux内核 4.4.0-142-generic （要求内核版本为4.4.0）

块设备（block device）是Linux三大设备之一，其驱动模型主要针对磁盘，Flash等存储类设备，块设备是一种具有一定结构的随机存取设备，对这种设备的读写是按块进行的，他使用缓冲区来存放暂时的数据，待条件成熟后，从缓存一次性写入设备或者从设备一次性读到缓冲区。

块设备核心结构：

（1）gendisk是一个磁盘或分区在内核中的描述。

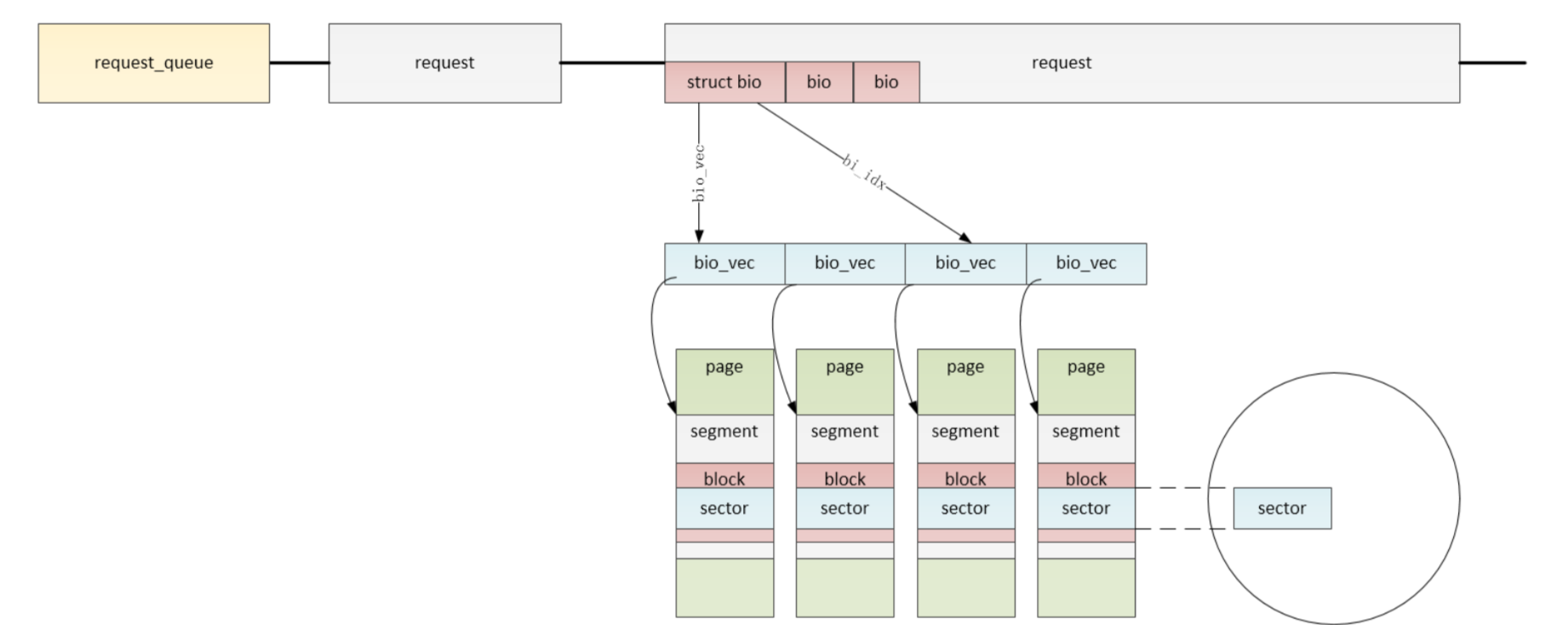
（2）block\_device\_operations描述磁盘的操作方法集。

（3）request\_queue每一个gendisk对象都有一个request\_queue对象，表示针对一个gendisk对象的所有请求的队列。

（4）request表示经过IO调度之后的针对一个gendisk(磁盘)的一个“请求"，是request\_queue的一个节点。多个request构成了一个request\_queue。

（5）bio 描述符，通用块的核心数据结构，描述了块设备的 I/O 操作。

（6）bio\_vec描述 bio 中的每个段，多个bio\_vec形成一个bio，多个bio经过IO调度和合并之后可以形成一个request。



## 一、安装并切换Linux内核至指定版本

开发，编译，调试内核模块需要先准备内核开发环境，不同 Linux 内核版本的通用 block 层API 有很大变化，我们的my\_block块设备驱动程序基于Linux 4.4.0开发，所以首先要在Ubuntu 20.04 LTS虚拟机中安装Linux内核 4.4.0-142-generic。（不要使用已经安装好对应内核版本的虚拟机，因为给虚拟机安装指定内核，配置内核开发环境，也是一项考察内容。）

1. 添加内核安装源

#切换至root用户

su

#查看当前系统内核版本

uname -a

# 打开apt源文件

vim /etc/apt/sources.list

# 在文件末尾加入

deb http://security.ubuntu.com/ubuntu trusty-security main

# 更新配置

apt-get update

1. 安装新的内核

# 查看可以安装哪些内核版本

apt-cache search linux | grep linux-image

# 安装4.4.0-142

apt install linux-image-extra-4.4.0-142-generic

# 查看是否安装成功

dpkg -l | grep 4.4.0-142-generic

1. 修改grub启动内核

# 修改grub文件

vim /etc/default/grub

找到GRUB\_DEFAULT=0这一行，我们把它改成 GRUB\_DEFAULT=”Ubuntu, Linux 4.4.0-142-generic” 然后保存。

# 更新grub启动文件

update-grub

1. 重启虚拟机系统

# 重启

reboot

# 查看内核版本是否变化

uname -a

如果显示4.4.0-142-generic，那么恭喜，内核切换成功，可以进入下一步，编写驱动程序了。

## 二、编写my\_block块设备驱动程序

1. 驱动程序要求

通过在内存中分配64MB的空间来模拟硬盘，按照块设备驱动编程的框架实现一个简单的块设备驱动程序，实现块设备的读写、格式化、挂接等操作。

2. 环境要求

在Ubuntu 20.04 LTS虚拟机中安装Linux内核 4.4.0-142-generic，并完成块设备驱动程序编译和测试。

3. 打开虚拟机系统终端，切换至root用户（以下操作都在root下完成），切换至根目录，创建实验项目文件夹

su

cd /

mkdir my\_block

cd my\_block

4. 编写驱动程序my\_block.c和Makefile

（1）导入头文件。

1. #include <linux/module.h>
2. #include <linux/blkdev.h>

（2）定义一些变量。

1. #define MY\_BLOCK\_NAME "my\_block"                // 块设备名
2. #define MY\_BLOCK\_MAJOR COMPAQ\_SMART2\_MAJOR      // 主设备号
3. #define MY\_BLOCK\_SIZE ((64)\*(1024)\*(1024))      // 块设备大小为64MB, 可以自定义
4. #define SECTOR\_SIZE\_SHIFT 9
6. **static** **struct** gendisk \*my\_block\_disk;   // gendisk结构表示一个简单的磁盘设备
7. **static** **struct** request\_queue \*my\_block\_queue;  // 指向块设备请求队列的指针
8. unsigned **char** my\_block\_data[MY\_BLOCK\_SIZE];   // 虚拟磁盘块设备的存储空间

（3）定义块设备操作my\_block\_fops，这是gendisk的一个属性。

1. **static** **int** my\_block\_ioctl(**struct** block\_device \*bdev, fmode\_t mode, unsigned command, unsigned **long** argument){
2. printk("ioctl cmd 0x%08x\n", command);
4. **return** -ENOTTY;
5. }
7. **static** **int** my\_block\_open(**struct** block\_device \*bdev, fmode\_t mode){
8. printk(">>> my\_block\_open\n");
10. **return** 0;
11. }
13. **static** **void** my\_block\_release(**struct** gendisk \*disk, fmode\_t mode){
14. printk(">>> my\_block\_release\n");
15. }
17. // 定义块设备操作my\_block\_fops，这是gendisk的一个属性
18. **static** **const** **struct** block\_device\_operations my\_block\_fops = {
19. .owner = THIS\_MODULE,
20. .open = my\_block\_open,
21. .release = my\_block\_release,
22. .ioctl = my\_block\_ioctl,
23. };

（4）模块入口函数，加载驱动。

1. 先申请块设备的资源，创建一个gendisk。
2. 初始化一个请求队列，绑定函数块设备请求处理函数my\_block\_do\_request。
3. 设置块设备的有关属性。
4. 注册块设备my\_block\_disk。
5. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
6. \*
7. \*   my\_block模块的入口函数
8. \*
9. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
10. **static** **int** \_\_init my\_block\_init(**void**){
11. **int** ret;
13. // 先申请块设备的资源
14. my\_block\_disk = alloc\_disk(1); // 使用 alloc\_disk 分配个 struct gendisk
15. **if**(!my\_block\_disk){
16. ret = -ENOMEM;
17. **goto** err\_alloc\_disk;
18. }
20. // 初始化一个请求队列，将块设备请求处理函数的地址传入blk\_init\_queue函数
21. my\_block\_queue = blk\_init\_queue(my\_block\_do\_request, NULL);
22. **if**(!my\_block\_queue){
23. ret = -ENOMEM;
24. **goto** err\_init\_queue;
25. }
27. // 设置块设备的有关属性
28. strcpy(my\_block\_disk->disk\_name, MY\_BLOCK\_NAME); // 设置设备名
29. my\_block\_disk->major = MY\_BLOCK\_MAJOR; // 主设备号，同一磁盘的各个分区共享一个主设备号。
30. my\_block\_disk->first\_minor = 0;  // 次设备号
31. my\_block\_disk->fops = my\_block\_fops; // 块设备操作函数指针fops
32. my\_block\_disk->queue = my\_block\_queue;
33. set\_capacity(my\_block\_disk, MY\_BLOCK\_SIZE>>9); // 设置块设备的大小，大小是扇区的数量，一个扇区是512B
35. // 注册块设备
36. add\_disk(my\_block\_disk);
38. printk("module %s load SUCCESS...\n", MY\_BLOCK\_NAME);
39. **return** 0;
41. err\_alloc\_disk:
42. blk\_cleanup\_queue(my\_block\_queue);
43. err\_init\_queue:
44. **return** ret;
45. }

（5）完成块设备请求的处理函数my\_block\_do\_request，实现读写功能。（本部分需要同学理解代码并补全）

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \*
3. \*   my\_block数据请求的处理函数
4. \*
5. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
6. **static** **void** my\_block\_do\_request(**struct** request\_queue \*q){
7. **struct** request \*req;  // 正在处理的请求队列中的请求
8. **struct** bio \*req\_bio;  // 当前请求的bio
9. **struct** bio\_vec \*bvec; // 当前请求的bio的段(segment)链表
10. **char** \*disk\_mem;       // 需要读/写的磁盘区域
11. **char** \*buffer;         // 磁盘块设备的请求在内存中的缓冲区
12. **int** i = 0;
14. **while**((req = blk\_fetch\_request(q)) != NULL){
15. // 判断当前request是否合法
16. **if**((blk\_rq\_pos(req)<<SECTOR\_SIZE\_SHIFT) + blk\_rq\_bytes(req) > MY\_BLOCK\_SIZE){
17. printk(KERN\_ERR MY\_BLOCK\_NAME":bad request:block=%llu, count=%u\n",(unsigned **long** **long**)blk\_rq\_pos(req),blk\_rq\_sectors(req));
18. blk\_end\_request\_all(req, -EIO);
19. **continue**;
20. }
21. //获取需要操作的内存位置
22. disk\_mem = my\_block\_data + (blk\_rq\_pos(req) << SECTOR\_SIZE\_SHIFT);
23. req\_bio = req->bio;// 获取当前请求的bio
25. **switch** (rq\_data\_dir(req)) {  //判断请求的类型，读还是写
26. **case** READ:
27. // 遍历request请求的bio链表
28. **while**(req\_bio != NULL){
29. //　for循环处理bio结构中的bio\_vec结构体数组（bio\_vec结构体数组代表一个完整的缓冲区）
30. **for**(i=0; i<req\_bio->bi\_vcnt; i++){
31. bvec = &(req\_bio->bi\_io\_vec[i]);
32. buffer = kmap(bvec->bv\_page) + bvec->bv\_offset; // 页数加偏移量获得对应的内存地址
33. memcpy(buffer, disk\_mem, bvec->bv\_len); // 将数据拷贝到内存中
34. kunmap(bvec->bv\_page); // 归还线性地址
35. disk\_mem += bvec->bv\_len;
36. }
37. req\_bio = req\_bio->bi\_next;
38. }
39. \_\_blk\_end\_request\_all(req, 0);
40. **break**;
41. **case** WRITE:
42. // 这里需要同学理解代码并补全
43. **default**:
44. // rq\_data\_dir(req) 为 1 bit， 所以没有default
45. **break**;
46. }
47. }
48. }

（6）模块出口函数，卸载驱动。

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \*
3. \*   my\_block模块的出口函数
4. \*
5. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
6. **static** **void** \_\_exit my\_block\_exit(**void**){
8. **if** (my\_block\_disk){
9. del\_gendisk(my\_block\_disk); // 注销磁盘块设备
10. put\_disk(my\_block\_disk);    // 释放磁盘， gendisk 对应的 kobject 引用计数变为零，彻底释放掉 gendisk
11. }
12. **if** (my\_block\_queue){
13. blk\_cleanup\_queue(my\_block\_queue); // 停止并释放块设备 IO 请求队列
14. }
16. printk("module %s exit SUCCESS...\n", MY\_BLOCK\_NAME);
17. }

（7）添加声明和模块信息。

1. module\_init(my\_block\_init); // 声明模块的入口
2. module\_exit(my\_block\_exit); // 声明模块的出口
4. MODULE\_LICENSE("GPL");
5. MODULE\_AUTHOR ("my\_block"); // 自定义模块作者信息
6. MODULE\_DESCRIPTION("the RAM disk block device"); //自定义模块描述信息
7. MODULE\_VERSION ("1.0");

（8）完成Makefile

1. ifeq ($(KERNELRELEASE),)
2. KDIR := /lib/modules/$(shell uname -r)/build
3. PWD := $(shell pwd)
4. modules:
5. $(MAKE) -C $(KDIR) M=$(PWD) modules
6. modules\_install:
7. $(MAKE) -C $(KDIR) M=$(PWD) modules\_install
8. clean:
9. rm -rf \*.o \*~ core .depend .\*.cmd \*.ko \*.mod.c .tmp\_versions \*.mod modules.order \*.symvers
10. .PHONY: modules modules\_install clean
11. **else**
12. obj-m := my\_block.o
13. endif

完成以上代码，一个基本的块设备驱动my\_block就已经完成了，接下来就是激动人心的测试环节了。

参考文档：

[1]. <https://www.cnblogs.com/xiaojiang1025/p/6500557.html>

[2]. <https://blog.csdn.net/yayong/article/details/51585490>

[3]. <https://blog.csdn.net/u012319493/article/details/85214863>

## 三、测试my\_block块设备驱动程序

现在my\_block项目文件夹下，应该有2个程序，my\_block.c 和Makefile，下面我们就可以编译并测试my\_block块设备驱动程序了。

1. 编译模块

#编译

make

#查看模块信息

modinfo my\_block.ko



1. 安装my\_block模块

# 安装my\_block

insmod my\_block.ko

# 查看模块

lsblk

# 格式化分区

mkfs -t ext4 /dev/my\_block

# 挂载my\_block

mkdir -p /mnt/my\_block

mount /dev/my\_block /mnt/my\_block/



1. 读写测试

# 写入测试

cp Makefile /mnt/my\_block

# 卸载my\_block

umount /mnt/my\_block

# 查看

ls -al /mnt/my\_block

# 挂载my\_block

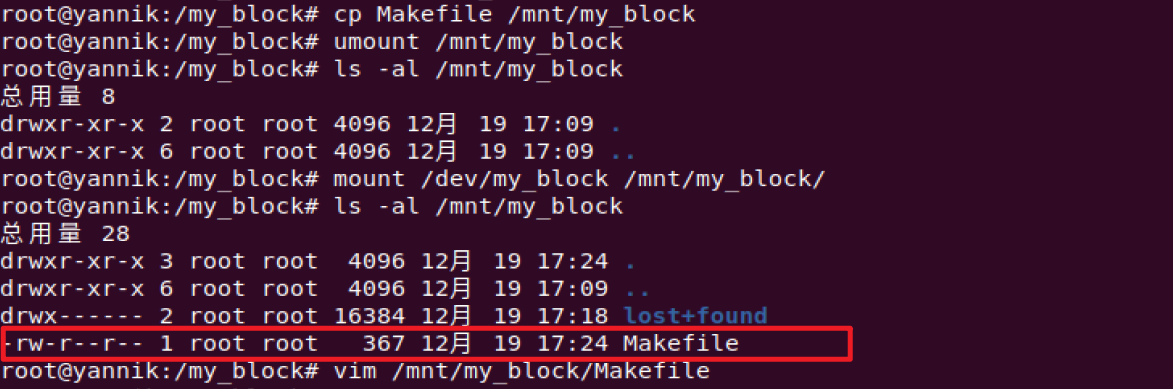
mount /dev/my\_block /mnt/my\_block/

# 查看Makefile是否在my\_block内

ls -al /mnt/my\_block

# 读取测试

vim /mnt/my\_block/Makefile



可以看到，给my\_block写入Makefile文件，卸载后Makefile消失，重新挂载后，读取Makefile文件正常。

1. 卸载并清理my\_block

# 卸载

umount /mnt/my\_block

# 卸载模块

rmmod my\_block

# 清理

make clean

至此，Linux块设备驱动程序实验全部完成，恭喜你！