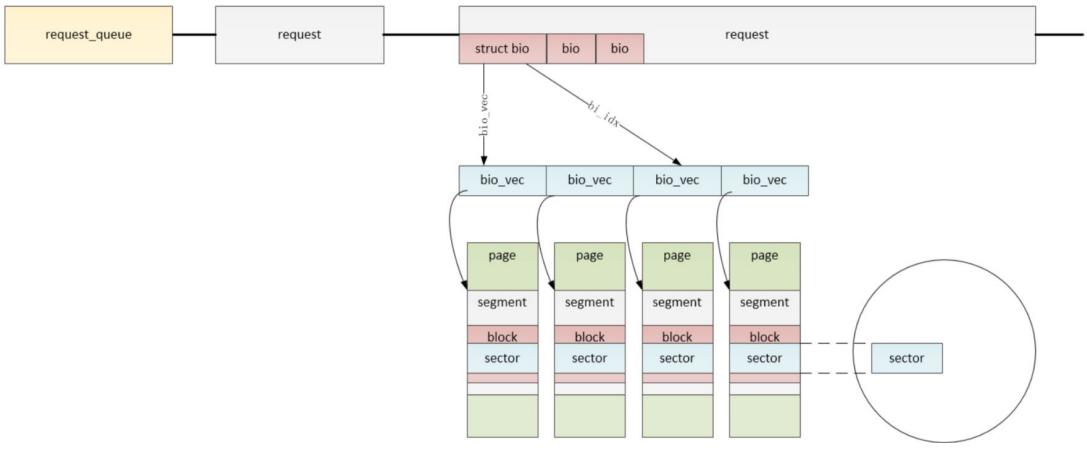
Linux 块设备驱动程序实验指导书

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
   ```

2. 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
   ```

3. 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
   ```

4. 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。

```
#include <linux/module.h>
#include <linux/blkdev.h>
```

（2）定义一些变量。

```
#define MY_BLOCK_NAME "my_block"                // 块设备名
#define MY_BLOCK_MAJOR COMPAQ_SMART2_MAJOR      // 主设备号
#define MY_BLOCK_SIZE ((64)*(1024)*(1024))      // 块设备大小为 64MB，可以自定义
#define SECTOR_SIZE_SHIFT 9

static struct gendisk *my_block_disk;   // gendisk 结构表示一个简单的磁盘设备
static struct request_queue *my_block_queue;  // 指向块设备请求队列的指针
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);
3.
4.      return -ENOTTY;
5.  }
6.
7.  static int my_block_open(struct block_device *bdev, fmode_t mode){
8.      printk(">>> my_block_open\n");
9.
10.     return 0;
11. }
12.
13. static void my_block_release(struct gendisk *disk, fmode_t mode){
14.     printk(">>> my_block_release\n");
15. }
16.
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。

a. 先申请块设备的资源，创建一个 gendisk。

b. 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。

c. 设置块设备的有关属性。

d. 注册块设备 my_block_disk。

```
1.  /*****************************************************
2.  *
3.  *   my_block 模块的入口函数
4.  *
5.  *****************************************************/
6.  static int __init my_block_init(void){
7.      int ret;
8.
9.      // 先申请块设备的资源
10.     my_block_disk = alloc_disk(1); // 使用 alloc_disk 分配个 struct gendisk
11.     if(!my_block_disk){
12.         ret = -ENOMEM;
13.         goto err_alloc_disk;
```

```
    }

    // 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
    my_block_queue = blk_init_queue(my_block_do_request, NULL);
    if(!my_block_queue){
        ret = -ENOMEM;
        goto err_init_queue;
    }

    // 设置块设备的有关属性
    strcpy(my_block_disk->disk_name, MY_BLOCK_NAME); // 设置设备名
    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。
    my_block_disk->first_minor = 0;  // 次设备号
    my_block_disk->fops = my_block_fops; // 块设备操作函数指针 fops
    my_block_disk->queue = my_block_queue;
    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

    // 注册块设备
    add_disk(my_block_disk);

    printk("module %s load SUCCESS...\n", MY_BLOCK_NAME);
    return 0;

    err_alloc_disk:
        blk_cleanup_queue(my_block_queue);
    err_init_queue:
        return ret;
}
```

（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）

```
/*******************************************************
*
*   my_block 数据请求的处理函数
*
*******************************************************/
static void my_block_do_request(struct request_queue *q){
    struct request *req;  // 正在处理的请求队列中的请求
    struct bio *req_bio;  // 当前请求的 bio
    struct bio_vec *bvec; // 当前请求的 bio 的段(segment)链表
    char *disk_mem;       // 需要读/写的磁盘区域
    char *buffer;         // 磁盘块设备的请求在内存中的缓冲区
```

```
12.     int i = 0;
13.
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_BLO
CK_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
24.
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。

```
/*****************************************************
*
*   my_block 模块的出口函数
*
*****************************************************/
static void __exit my_block_exit(void){

    if (my_block_disk){
        del_gendisk(my_block_disk); // 注销磁盘块设备
        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
    }
    if (my_block_queue){
        blk_cleanup_queue(my_block_queue); // 停止并释放块设备 IO 请求队列
    }

    printk("module %s exit SUCCESS...\n", MY_BLOCK_NAME);
}
```

（7）添加声明和模块信息。

```
module_init(my_block_init); // 声明模块的入口
module_exit(my_block_exit); // 声明模块的出口

MODULE_LICENSE("GPL");
MODULE_AUTHOR ("my_block"); // 自定义模块作者信息
MODULE_DESCRIPTION("the RAM disk block device"); //自定义模块描述信息
MODULE_VERSION ("1.0");
```

（8）完成 Makefile

```
ifeq ($(KERNELRELEASE),)
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
modules:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
modules_install:
    $(MAKE) -C $(KDIR) M=$(PWD) modules_install
clean:
    rm -
rf *.o *~ core .depend .*.cmd *.ko *.mod.c .tmp_versions *.mod modules.order *.symvers
.PHONY: modules modules_install clean
else
    obj-m := my_block.o
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

```
root@yannik:/my_block# make
make -C /lib/modules/4.4.0-142-generic/build M=/my_block modules
make[1]: 进入目录“/usr/src/linux-headers-4.4.0-142-generic”
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: 离开目录“/usr/src/linux-headers-4.4.0-142-generic”
root@yannik:/my_block# modinfo my_block.ko
filename:       /my_block/my_block.ko
version:        1.0
description:    the RAM disk block device
author:         my_block
license:        GPL
srcversion:     9993912AD93A4CF6B1C4F69
depends:
retpoline:      Y
vermagic:       4.4.0-142-generic SMP mod_unload modversions retpoline
```

2. 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/

```
root@yannik:/my_block# insmod my_block.ko
root@yannik:/my_block# lsblk
NAME     MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda        8:0    0   60G  0 disk
└─sda1     8:1    0   60G  0 part /
sr0       11:0    1 1024M  0 rom
my_block  72:0    0   64M  0 disk
root@yannik:/my_block# mkfs -t ext4 /dev/my_block
mke2fs 1.45.5 (07-Jan-2020)
创建含有 16384 个块（每块 4k）和 16384 个inode的文件系统

正在分配组表：  完成
正在写入inode表：  完成
创建日志（1024 个块）  完成
写入超级块和文件系统账户统计信息：  已完成

root@yannik:/my_block# mkdir -p /mnt/my_block
root@yannik:/my_block# mount /dev/my_block /mnt/my_block/
```

3. 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

```
root@yannik:/my_block# cp Makefile /mnt/my_block
root@yannik:/my_block# umount /mnt/my_block
root@yannik:/my_block# ls -al /mnt/my_block
总用量 8
drwxr-xr-x 2 root root 4096 12月  19 17:09 .
drwxr-xr-x 6 root root 4096 12月  19 17:09 ..
root@yannik:/my_block# mount /dev/my_block /mnt/my_block/
root@yannik:/my_block# ls -al /mnt/my_block
总用量 28
drwxr-xr-x 3 root root  4096 12月  19 17:24 .
drwxr-xr-x 6 root root  4096 12月  19 17:09 ..
drwx------ 2 root root 16384 12月  19 17:18 lost+found
-rw-r--r-- 1 root root   367 12月  19 17:24 Makefile
root@yannik:/my_block# 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。

4. 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!