МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное
учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО
ПРИБОРОСТРОЕНИЯ»
Кафедра 43

КУРСОВАЯ РАБОТА
ЗАЩИЩЕНА С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ
ст.преп _____ М.Д.Поляк

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ

Создание виртуального устройства

по дисциплине: ОПЕРАЦИОННЫЕ СИСТЕМЫ И ОБОЛОЧКИ

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР. 4336 _____ К.С.Бутузов

Санкт-Петербург, 2016 г.

1 Цель работы

Знакомство с устройством ядра ОС Linux. Получение опыта разработки драйвера устройства.

2 Индивидуальное задание

Создание виртуального устройства. Реализовать драйвер для создания RAM-диска.

Проверка правильности функционирования драйвера включает в себя создание RAM-диска, его форматирование в одну из популярных файловых систем, копирование файлов на диск и с диска, удаление RAM-диска.

3 Сравнение с аналогами

Как известно, RAM-диск – это программная технология, позволяющая хранить данные в быстродействующей оперативной памяти как на блочном устройстве. Обычно является составной частью операционной системы. В ряде случаев – это программа стороннего производителя.

Основные достоинства — высокая скорость чтения (измеряется гигабайтами в секунду), высокие показатели IOPS (операций ввода-вывода в секунду) — некоторые образцы оперативной памяти типа DDR3 позволяют достигать более 1 млн IOPS (у дисковых накопителей — 20—300 IOPS, NAND SSD — десятки—сотни тысяч IOPS), отсутствие дополнительных задержек при произвольном доступе, неограниченный ресурс перезаписи (в отличие от флеш-памяти). Среди недостатков — относительно малые ёмкости модулей оперативной памяти, потеря содержимого при отключении питания, высокая стоимость за гигабайт.

Linux реализует следующие виды RAM-дисков:

-специализированный архив в формате cpio для размещения модулей для начальной загрузки (initrd);

-файловая система, размещающаяся в памяти tmpfs (используется чаще всего для хранения временных данных, сохранение которых не актуально между перезагрузками и к которым нужен быстрый доступ);

-модуль brd, позволяющий создавать блочные устройства (вида /dev/ram0);

-модуль zram, позволяющий создавать блочные устройства вида /dev/zram0, хранящий данные в памяти в сжатом виде.

В задании требуется написать драйвер для создания RAM-диска, а также после его создания произвести с ним простейшие операции – его форматирование в одну из популярных файловых систем, копирование файлов на диск и с диска, удаление RAM-диска. Поэтому наиболее адекватным вариантов в нашем случае будет написать модуль brd.

Создание RAM-диска – достаточно стандартная и популярная операция, поэтому вариантов реализации в принципе немного, а отличаются они, как правило, размерами RAM-диска и секторов.

4 Техническая документация

4.1 Сборка модуля

Сначала необходимо собрать модуль ядра. Для этого написан Makefile. Сборку можно выполнить командой make. На некоторых устройствах нужно подробно прописывать путь, например: make -C/lib/modules/3.16.0-4-686-pae/build SUBDIRS=/home/konstant19961996/RAM-disk. Иногда нужно указать путь к исходникам ядра и к папке, где содержатся файлы

4.2 Загрузка модуля в ядро

Модуль загружается в ядро командой insmod dor.ko (где dor.ko - название модуля)

4.3 Посмотреть структуру диска

Проще всего посмотреть структуру диска при помощи команды lsblk

4.4 Отформатировать раздел

Отформатировать раздел можно с помощью команды mkfs.vfat /dev/rb1

4.5 Смонтировать раздел

Смонтировать отформатированный раздел можно командой mount -t vfat /dev/rb1 /mnt/

4.6 Записать файл в раздел

Записать файл в раздел можно при помощи команды cp. При этом стоит смотреть на размер файла, который вы хотите записать

4.7 Размонтировать раздел

Размонтировать раздел можно при помощи команды umount /mnt

4.8 Выгрузить модуль

Выгрузка модуля осуществляется командой rmmod

5 Выводы

В ходе выполнения курсовой работы был написан драйвер для создания виртуального устройства, который позволяет ускорить работу с памятью в необходимых случаях. Для этого была изучена организация памяти в ОС Linux, поскольку данный драйвер – блочного устройства. Был изучен процесс написания модулей ядра под Linux, загрузка их в ядро, а также последующая работа с ними.

6 Приложение

6.1 Файл part.h

```
#ifndef PART_H
    #define PART_H
    #include <linux/types.h>
    extern void copy_mbr_to_br(u8 *disk);
```

6.2 Файл part.c

```
#include <linux/string.h>
    #include "part.h"
    #define ARRAY_SIZE(a) (sizeof(a) / sizeof(*a))
    #define SECTOR_SIZE 512
    #define MBR_SIZE SECTOR_SIZE
    #define MBR_DISK_SIGNATURE_OFFSET 440
    #define MBR_DISK_SIGNATURE_SIZE 4
    #define PARTITION_TABLE_OFFSET 446
    #define PARTITION_ENTRY_SIZE 16
    #define PARTITION_TABLE_SIZE 64
    #define MBR_SIGNATURE_OFFSET 510
    #define MBR_SIGNATURE_SIZE 2
    #define MBR_SIGNATURE 0xAA55
    #define BR_SIZE SECTOR_SIZE
    #define BR_SIGNATURE_OFFSET 510
    #define BR_SIGNATURE_SIZE 2
    #define BR_SIGNATURE 0xAA55
    typedef struct
    { unsigned char boottype;
    unsigned char starthead;
    unsigned char startsec:6;
    unsigned char startcyl_hi:2;
    unsigned char startcyl;
    unsigned char parttype;
    unsigned char endhead;
    unsigned char endsec:6;
    unsigned char endcyl_hi:2;
    unsigned char endcyl;
    unsigned int abssecstart;
    unsigned int secpart;
```

```
} PartEntry;
typedef PartEntry PartTable[4];
static PartTable def_part_table =
{
{
boottype: 0x00,
starthead: 0x00,
startsec: 0x2,
startcyl: 0x00,
parttype: 0x83,
endhead: 0x00,
endsec: 0x20,
endcyl: 0x09,
abssecstart: 0x00000001,
secpart: 0x0000013F
},
{
boottype: 0x00,
starthead: 0x00,
startsec: 0x1,
startcyl: 0x0A,
parttype: 0x05,
endhead: 0x00,
endsec: 0x20,
endcyl: 0x13,
abssecstart: 0x00000140,
secpart: 0x00000140
},
{
boottype: 0x00,
starthead: 0x00,
startsec: 0x1,
startcyl: 0x14,
parttype: 0x83,
endhead: 0x00,
endsec: 0x20,
endcyl: 0x1F,
abssecstart: 0x00000280,
secpart: 0x00000180
},
```

```
{
}
};
static unsigned int def_log_part_br_cyl[] = {0x0A, 0x0E, 0x12};
static const PartTable def_log_part_table[] =
{
{
{
boottype: 0x00,
starthead: 0x00,
startsec: 0x2,
startcyl: 0x0A,
parttype: 0x83,
endhead: 0x00,
endsec: 0x20,
endcyl: 0x0D,
abssecstart: 0x00000001,
secpart: 0x0000007F
},
{
boottype: 0x00,
starthead: 0x00,
startsec: 0x1,
startcyl: 0x0E,
parttype: 0x05,
endhead: 0x00,
endsec: 0x20,
endcyl: 0x11,
abssecstart: 0x00000080,
secpart: 0x00000080
},
},
{
{
boottype: 0x00,
starthead: 0x00,
startsec: 0x2,
startcyl: 0x0E,
parttype: 0x83,
endhead: 0x00,
```

```
endsec: 0x20,
endcyl: 0x11,
abssecstart: 0x00000001,
secpart: 0x0000007F
},
{
boottype: 0x00,
starthead: 0x00,
startsec: 0x1,
startcyl: 0x12,
parttype: 0x05,
endhead: 0x00,
endsec: 0x20,
endcyl: 0x13,
abssecstart: 0x00000100,
secpart: 0x00000040
},
},
{
{
boottype: 0x00,
starthead: 0x00,
startsec: 0x2,
startcyl: 0x12,
parttype: 0x83,
endhead: 0x00,
endsec: 0x20,
endcyl: 0x13,
abssecstart: 0x00000001,
secpart: 0x0000003F
},
}
};
static void copymbr(u8 *disk)
{
memset(disk, 0x0, MBR_SIZE);
*(unsigned long *)(disk + MBR_DISK_SIGNATURE_OFFSET) = 0x36E5756D;
memcpy(disk + PARTITION_TABLE_OFFSET, def_part_table, PARTITION_TABLE_SIZ
*(unsigned short *)(disk + MBR_SIGNATURE_OFFSET) = MBR_SIGNATURE;
}
```

```
static void copybr(u8 *disk, int start_cylinder, const PartTable *part_table)
{
disk += (start_cylinder * 32 * SECTOR_SIZE);
memset(disk, 0x0, BR_SIZE);
memcpy(disk + PARTITION_TABLE_OFFSET, part_table, PARTITION_TABLE_SIZE);
*(unsigned short *)(disk + BR_SIGNATURE_OFFSET) = BR_SIGNATURE;
}
void copy_mbr_ti_br(u8 *disk)
{ int i;
copymbr(disk);
for (i = 0; i < ARRAY_SIZE(def_log_part_table); i++)
{
copybr(disk, def_log_part_br_cyl[i], def_log_part_table[i]);
}
}
```

6.3 Файл ramdevice.h

```
#ifndef RAMDEVICE_H
    #define RAMDEVICE_H
    #define RB_SECTOR_SIZE 512
    extern int ramdevice_initial(void);
    extern void ramdevice_clean(void);
    extern void ramdevice_write(sector_t sector_off, u8 *buffer, unsigned
int sectors);
    extern void ramdevice_read(sector_t sector_off, u8 *buffer, unsigned
int sectors);
    #endif
```

6.4 Файл ramdevice.c

```
#include <linux/types.h>
    #include <linux/vmalloc.h>
    #include <linux/string.h>
    #include <linux/errno.h>
    #include "ramdevice.h"
    #include "part.h"
    #define RB_DEVICE_SIZE 1024
    static u8 *dev_data;
    int ramdevice_initial(void)
```

```
    {
    dev_data = vmalloc(RB_DEVICE_SIZE * RB_SECTOR_SIZE);
    if (dev_data == NULL)
    return -ENOMEM; copy_mbr_to_br(dev_data);
    return RB_DEVICE_SIZE;
    }
    void ramdevice_clean(void)
    {
    vfree(dev_data);
    }
    void ramdevice_write(sector_t sector_off, u8 *buffer, unsigned int sectors)
    {
    memcpy(dev_data + sector_off * RB_SECTOR_SIZE, buffer, sectors
* RB_SECTOR_SIZE);
    }
    void ramdevice_read(sector_t sector_off, u8 *buffer, unsigned int sectors)
    {
    memcpy(buffer, dev_data + sector_off * RB_SECTOR_SIZE, sectors
* RB_SECTOR_SIZE);
    }
```

6.5 Файл ramblock.c

```
#include <linux/module.h>
    #include <linux/kernel.h>
    #include <linux/version.h>
    #include <linux/fs.h>
    #include <linux/types.h>
    #include <linux/spinlock.h>
    #include <linux/genhd.h>
    #include <linux/blkdev.h>
    #include <linux/hdreg.h>
    #include <linux/errno.h>
    #include "ramdevice.h"
    #define RB_FIRST_MINOR 0
    #define RB_MINOR_CNT 16
    static u_int rb_major = 0;
    static struct rbdevice
    {
    unsigned int size;
```

```
spinlock_t lock;
struct request_queue *rb_queue;
struct gendisk *rb_disk;
} rb_dev;
static int rbopen(struct block_device *bdev, fmode_t mode)
{
unsigned unit = iminor(bdev->bd_inode);
printk(KERN_INFO "rb: Device is opened");
printk(KERN_INFO "rb: Inode number is %d unit);
if (unit > RB_MINOR_CNT)
return -ENODEV;
return 0;
}
#if (LINUX_VERSION_CODE < KERNEL_VERSION(3,10,0))
static int rbclose(struct gendisk *disk, fmode_t mode)
{
printk(KERN_INFO "rb: Device is closed");
return 0;
}
#else
static void rbclose(struct gendisk *disk, fmode_t mode)
{
printk(KERN_INFO "rb: Device is closed");
}
#endif
static int rbgetgeo(struct block_device *bdev, struct hd_geometry *geo)
{
geo->heads = 1;
geo->cylinders = 32;
geo->sectors = 32;
geo->start = 0;
return 0;
}
static int rbtransfer(struct request *req)
{
int dir = rq_data_dir(req);
sector_t start_sector = blk_rq_pos(req);
unsigned int sector_cnt = blk_rq_sectors(req);
#if (LINUX_VERSION_CODE < KERNEL_VERSION(3,14,0))
#define BV_PAGE(bv) ((bv)->bv_page)
```

```
    #define BV_OFFSET(bv) ((bv)->bv_offset)
    #define BV_LEN(bv) ((bv)->bv_len)
    struct bio_vec *bv;
    #else
    #define BV_PAGE(bv) ((bv).bv_page)
    #define BV_OFFSET(bv) ((bv).bv_offset)
    #define BV_LEN(bv) ((bv).bv_len)
    struct bio_vec bv;
    #endif
    struct req_iterator iter;
    sector_t sector_offset;
    unsigned int sectors;
    u8 *buffer;
    int ret = 0;
    sector_offset = 0;
    rq_for_each_segment(bv, req, iter)
    {
    buffer = page_address(BV_PAGE(bv)) + BV_OFFSET(bv);
    if (BV_LEN(bv) % RB_SECTOR_SIZE != 0)
    {
    printk(KERN_ERR "rb: Should never happen: "
    "bio size (%d) is not a multiple of RB_SECTOR_SIZE (%d)."
    "This may lead to data truncation.
    BV_LEN(bv), RB_SECTOR_SIZE);
    ret = -EIO;
    }
    sectors = BV_LEN(bv) / RB_SECTOR_SIZE;
    printk(KERN_DEBUG "rb: Start Sector: %llu, Sector Offset: %llu; Buffer:
%p; Length: %u sectors
    (unsigned long long)(start_sector), (unsigned long long)(sector_offset),
buffer, sectors);
    if (dir == WRITE)
    {
    ramdevice_write(start_sector + sector_offset, buffer, sectors);
    }
    else
    {
    ramdevice_read(start_sector + sector_offset, buffer, sectors);
    }
    sector_offset += sectors;
```

```
}
if (sector_offset != sector_cnt)
{
printk(KERN_ERR "rb: bio info doesn't match with the request info");
ret = -EIO;
}
return ret;
}
static void rbrequest(struct request_queue *q)
{
struct request *req;
int ret;
while ((req = blk_fetch_request(q)) != NULL)
{
#if 0
if (!blk_fs_request(req))
{
printk(KERN_NOTICE "rb: Skip non-fs request");
__blk_end_request_all(req, 0);
continue;
}
#endif
ret = rb_transfer(req);
__blk_end_request_all(req, ret);
}
}
static struct block_device_operations rb_fops =
{
.owner = THIS_MODULE,
.open = rb_open,
.release = rb_close,
.getgeo = rb_getgeo,
};
static int __init rbinit(void)
{
int ret;
if ((ret = ramdevice_init()) < 0)
{
return ret;
}
```

```
    rb_dev.size = ret;
    rb_major = register_blkdev(rb_major, "rb");
    if (rb_major <= 0)
    {
    printk(KERN_ERR "rb: Unable to get Major Number");
    ramdevice_cleanup();
    return -EBUSY;
    }
    spin_lock_init(rb_dev.lock);
    rb_dev.rb_queue = blk_init_queue(rb_request, rb_dev.lock);
    if (rb_dev.rb_queue == NULL)
    {
    printk(KERN_ERR "rb: blk_init_queue failure");
    unregister_blkdev(rb_major, "rb");
    ramdevice_cleanup();
    return -ENOMEM;
    }
    rb_dev.rb_disk = alloc_disk(RB_MINOR_CNT) ; if (!rb_dev.rb_disk)
    {
    printk(KERN_ERR "rb: alloc_disk failure");
    blk_cleanup_queue(rb_dev.rb_queue);
    unregister_blkdev(rb_major, "rb");
    ramdevice_cleanup();
    return -ENOMEM;
    }
    rb_dev.rb_disk->major = rb_major;
    rb_dev.rb_disk->first_minor = RB_FIRST_MINOR;
    rb_dev.rb_disk->fops = rb_fops;
    rb_dev.rb_disk->private_data = rb_dev;
    rb_dev.rb_disk->queue = rb_dev.rb_queue;
    sprintf(rb_dev.rb_disk->disk_name, "rb");
    set_capacity(rb_dev.rb_disk, rb_dev.size);
    add_disk(rb_dev.rb_disk);
    printk(KERN_INFO "rb: Ram Block driver initialised (%d sectors; %d
bytes) rb_dev.size, rb_dev.size * RB_SECTOR_SIZE);
    return 0;
    }
    static void __exit rbcleanup(void)
    {
    del_gendisk(rb_dev.rb_disk);
```

```
    put_disk(rb_dev.rb_disk);
    blk_cleanup_queue(rb_dev.rb_queue);
    unregister_blkdev(rb_major, "rb");
    ramdevice_cleanup();
    }
    module_init(rb_init);
    module_exit(rb_cleanup);
    MODULE_LICENSE("GPL");
    MODULE_AUTHOR("Konstantin Butuzov <bks1906@gmail.com>");
    MODULE_DESCRIPTION("Ram Block Driver");
    MODULE_ALIAS_BLOCKDEV_MAJOR(rb_major);
```

6.6 Makefile

```
ifeq ($(KERNELRELEASE),)
    KERNEL_SOURCE := /lib/modules/$(shell uname -r)/build
    PWD := $(shell pwd)
    module:
    $(MAKE) -C $(KERNEL_SOURCE) SUBDIRS=$(PWD) modules
    clean:
    $(MAKE) -C $(KERNEL_SOURCE) SUBDIRS=$(PWD) clean
    else
    obj-m := dor.o
    dor-y := ramblock.o ramdevice.o part.o
    endif
```