《操作系统原理》实验报告

姓名：韩艺轩
学院：计算机科学与技术学院
专业：计算机科学与技术（图灵班）
邮箱：2674361965@qq.com
指导教师：申文博

报告日期: 2024/01/10

Lab7 - VFS & FAT32 文件系统

本次实验完成了全部的bnous。

实验步骤

Shell: 与内核进行交互

完善`syscall.c`

将read，lseek，close,openat等系统调用都完善。

```
#include "syscall.h"
#include "fs.h"
#include "defs.h"
#include "types.h"
#include "virtio.h"
#include "fat32.h"

extern struct task_struct* current;

int64_t sys_read(unsigned int fd, char* buf, uint64_t count) {
    int64_t ret;
    struct file* target_file = &(current->files[fd]);
    if (target_file->opened) {
        ret = target_file->read(target_file, buf, count);
    } else {
        printk("file not open\n");
        ret = ERROR_FILE_NOT_OPEN;
    }
    return ret;
}
int64_t sys_write(unsigned int fd, const char* buf, uint64_t count) {
    int64_t ret;
    struct file* target_file = &(current->files[fd]);
    if (target_file->opened) {
        ret = target_file->write(target_file, buf, count);
    } else {
        printk("file not open\n");
        ret = ERROR_FILE_NOT_OPEN;
    }
    return ret;
}
int64_t sys_openat(int dfd, const char* filename, int flags) {
    int fd = -1;
    // printk("in sys_openat\n");
    // Find an available file descriptor first
    for (int i = 0; i < PGSIZE / sizeof(struct file); i++) {
        if (!current->files[i].opened) {
            fd = i;
            break;
        }
    }

    // Do actual open
    file_open(&(current->files[fd]), filename, flags);

    return fd;
}
int64_t sys_lseek(int fd, int64_t offset, int whence) {
    // printk("in sys_lseek\n");
    int64_t ret;
    struct file* target_file = &(current->files[fd]);
    if (target_file->opened) {
        ret = target_file->lseek(target_file, offset, whence);
    } else {
        printk("file not open\n");
        ret = ERROR_FILE_NOT_OPEN;
    }
    return ret;
}
void sys_close(int fd){
    struct file * target_file = &(current->files[fd]);
    if(target_file->opened){
        target_file->opened = 0;
        target_file->cfo = 0;
    }else{
        printk("file not open\n");
    }
}
void syscall(struct pt_regs *regs){
    regs->sepc += 4;
    if (regs->reg[17] == SYS_WRITE){
        regs->reg[10] = sys_write(regs->reg[10], (const char *)regs->reg[11], regs->reg[12]);
    }else if(regs->reg[17] == SYS_GETPID){
        regs->reg[10] = current->pid;
    }else if(regs->reg[17] == SYS_READ){
        regs->reg[10] = sys_read(regs->reg[10], regs->reg[11], regs->reg[12]);
    }else if(regs->reg[17] == SYS_OPENAT){
        regs->reg[10] = sys_openat(regs->reg[10], regs->reg[11], regs->reg[12]);
    }else if(regs->reg[17] == SYS_LSEEK){
        regs->reg[10] = sys_lseek(regs->reg[10], regs->reg[11], regs->reg[12]);
    }else if(regs->reg[17] == SYS_CLOSE){
        sys_close(regs->reg[10]);
    }
}
```

初始化工作

更新`task_struct`,添加一个files。

```
struct task_struct {
    struct thread_info thread_info;
    uint64 state;    // 线程状态
    uint64 counter;  // 运行剩余时间
    uint64 priority; // 运行优先级 1最低 10最高
    uint64 pid;      // 线程id

    struct thread_struct thread;
    pagetable_t pgd;
    struct file *files;
};
```

在进程刚刚被创建的时候，先完成打开的文件的列表的初始化。

```
for(uint64 i = 1;i<NR_TASKS;i++){
        task[i] = (struct task_struct *)kalloc();
        task[i]->files = file_init();
        .......
}
```

在`vfs.c`中完成`task_init()`.

```
struct file* file_init() {
    struct file *ret = (struct file*)alloc_page();
    // stdin
    ret[0].opened = 1;
    ret[0].perms = FILE_READABLE;
    ret[0].cfo = 0;
    ret[0].lseek = NULL;
    ret[0].write = NULL;
    ret[0].read = stdin_read;
    memcpy(ret[0].path, "stdin", 6);

    // ...

    // stdout
    ret[1].opened = 1;
    ret[1].perms = FILE_WRITABLE;
    ret[1].cfo = 0;
    ret[1].lseek = NULL;
    ret[1].write = stdout_write;
    ret[1].read = NULL;
    memcpy(ret[1].path, "stdout", 7);

    // stderr
    ret[2].opened = 1;
    ret[2].perms = FILE_WRITABLE | FILE_READABLE;
    ret[2].cfo = 0;
    ret[2].lseek = NULL;
    ret[2].write = stderr_write;
    ret[2].read = NULL;
    memcpy(ret[2].path, "stderr", 7);
    // ...
    return ret;
}
```

其中有三个函数指针，所指的函数实现如下：

```
int64_t stdin_read(struct file* file, void* buf, uint64_t len) {
    /* todo: use uart_getchar() to get <len> chars */
    char *read = (char *)buf;
    for(int i = 0;i < len; i++){
        read[i] = uart_getchar();
    }
    return len;
}

int64_t stdout_write(struct file* file, const void* buf, uint64_t len) {
    char to_print[len + 1];
    for (int i = 0; i < len; i++) {
        to_print[i] = ((const char*)buf)[i];
    }
    to_print[len] = 0;
    return printk(buf);
}

int64_t stderr_write(struct file* file, const void* buf, uint64_t len) {
    char to_print[len + 1];
    for (int i = 0; i < len; i++) {
        to_print[i] = ((const char*)buf)[i];
    }
    to_print[len] = 0;
    return printk(buf);
}
```

至此，可以实现与shell的交互，结果在最后给出。

FAT32: 持久存储

对 VirtIO 和 MBR 进行初始化

```
void task_init() {
    ......
    printk("...proc_init done!\n");
    virtio_dev_init();
    mbr_init();
}
```

初始化 FAT32 分区

fat32_volume 是用来存储我们实验中需要用到的元数据的，需要根据 fat32_bpb 中的数据来进行计算并初始化。

```
void fat32_init(uint64_t lba, uint64_t size) {
    virtio_blk_read_sector(lba, (void*)&fat32_header);
    uint64 reserve = fat32_header.rsvd_sec_cnt;
    uint64 Fat = fat32_header.num_fats * fat32_header.fat_sz32;
    fat32_volume.first_data_sec = lba + reserve+Fat;
    fat32_volume.sec_per_cluster = fat32_header.sec_per_clus;
    fat32_volume.first_fat_sec = lba + reserve;
    fat32_volume.fat_sz = fat32_header.fat_sz32;

    virtio_blk_read_sector(fat32_volume.first_data_sec, fat32_buf); // Get the
root directory
    struct fat32_dir_entry *dir_entry = (struct fat32_dir_entry *)fat32_buf;
}
```

读取 FAT32 文件

大部分逻辑框架已经给出，只需要实现`fat32_open_file`.首先要截取文件名除去根目录，然后将文件名转换为大写，然后在根目录中找，获得簇号。

```
struct fat32_file fat32_open_file(const char *path) {
    // printk("in 32_file_open\n");
    char path_[strlen(path)];
    for(uint64 i = 0;i < strlen(path);i++){
        path_[i] = path[i+1];
    }
    uint64 temp = next_slash(path_);
    char t[strlen(path_)-temp-1];
    if(temp!=-1){
        for(uint64 i = 0;i<strlen(path_)-temp;i++){
            t[i] = path_[i+temp+1];
        }
    }
    struct fat32_file file;
    to_upper_case((char *)t);
    // printk("%d\n",strlen(t));
    virtio_blk_read_sector(fat32_volume.first_data_sec, fat32_buf); // Get the
root directory
    struct fat32_dir_entry *dir_entry = (struct fat32_dir_entry *)fat32_buf;
    uint64 i = 0;
    for(i=0;i<512;i++){
        // printk("dir_entry[i].name=%s",dir_entry[i].name);
        if(memcmp(dir_entry[i].name, t, strlen(t))==0){
            // printk("find!%d\n", i);
            file.cluster = dir_entry[i].startlow | dir_entry[i].starthi << 16;
            file.dir.cluster = dir_entry[i].startlow | dir_entry[i].starthi << 16;
            uint64 temp = cluster_to_sector(file.dir.cluster);
            file.dir.index = i;
            break;
        }
    }
    // printk("leaving...\n");
    return file;
}
```

实现 lseek syscall

`sys_lseek()`在报告一开始已经给出。不做赘述，按照不同的whence调整cfo的位置。

```
int64_t fat32_lseek(struct file* file, int64_t offset, uint64_t whence) {
    // printk("in lseek\n");
    if (whence == SEEK_SET) {
        file->cfo = offset;
    } else if (whence == SEEK_CUR) {
        file->cfo = file->cfo + offset;
    } else if (whence == SEEK_END) {
        /* Calculate file length */
        virtio_blk_read_sector(fat32_volume.first_data_sec, fat32_buf); // Get the root directory
        struct fat32_dir_entry *dir_entry = (struct fat32_dir_entry *)fat32_buf;
        file->cfo = dir_entry[file->fat32_file.dir.index].size;
    } else {
        printk("fat32_lseek: whence not implemented\n");
        while (1);
    }
    return file->cfo;
}
```

fat32_read()的实现

实现`fat32_read()`函数，因为每次只能最多读长度为 509 的字符串，所以需要分多次读入，靠file->cfo来记录读到哪里了，因为只能根据簇号读这一个簇中的扇区，所以在读完某一个簇的时候还需要重新获得下一个簇的扇区号。

```
int64_t fat32_read(struct file* file, void* buf, uint64_t len) {
    // printk("%d\n", file->cfo);
    virtio_blk_read_sector(fat32_volume.first_data_sec, fat32_buf); // Get the root directory
    struct fat32_dir_entry *dir_entry = (struct fat32_dir_entry *)fat32_buf;
    uint64 file_total_len = dir_entry[file->fat32_file.dir.index].size;
    // printk("total len = %d\n", file_total_len);
    uint64 cluster_size = fat32_volume.sec_per_cluster * 512;
    uint64 left_len = len;
    if(file_total_len - file->cfo < len){
        left_len = file_total_len - file->cfo;
    }
    uint64 ret = 0;
    uint32_t cluster = file->fat32_file.cluster + file->cfo/cluster_size;
    while(left_len > 0){
        uint64 sec = cluster_to_sector(cluster);
        virtio_blk_read_sector(sec, fat32_buf);
        uint64 offset = file->cfo % cluster_size;
        uint64 left_part_in_cluster = cluster_size - offset;
        // printk("offset = %d\n", offset);
        // printk("left_part = %d\n", left_part_in_cluster);
        // printk("left_len = %d\n", left_len);
        if(left_len >= left_part_in_cluster){
            memcpy(buf, fat32_buf + offset, left_part_in_cluster);
            file->cfo += left_part_in_cluster;
            left_len -= left_part_in_cluster;
            ret += left_part_in_cluster;
        }else{
            memcpy(buf, fat32_buf + offset, left_len);
            file->cfo += left_len;
            ret += left_len;
            left_len = 0;
        }
        uint32_t next_cluster_number = next_cluster(cluster);

        if (next_cluster_number >= 0x0ffffff8) {
            break;
        }
        cluster = next_cluster_number;
    }
    // printk("ret = %d", ret);
    return ret;
    /* todo: read content to buf, and return read length */
}
```

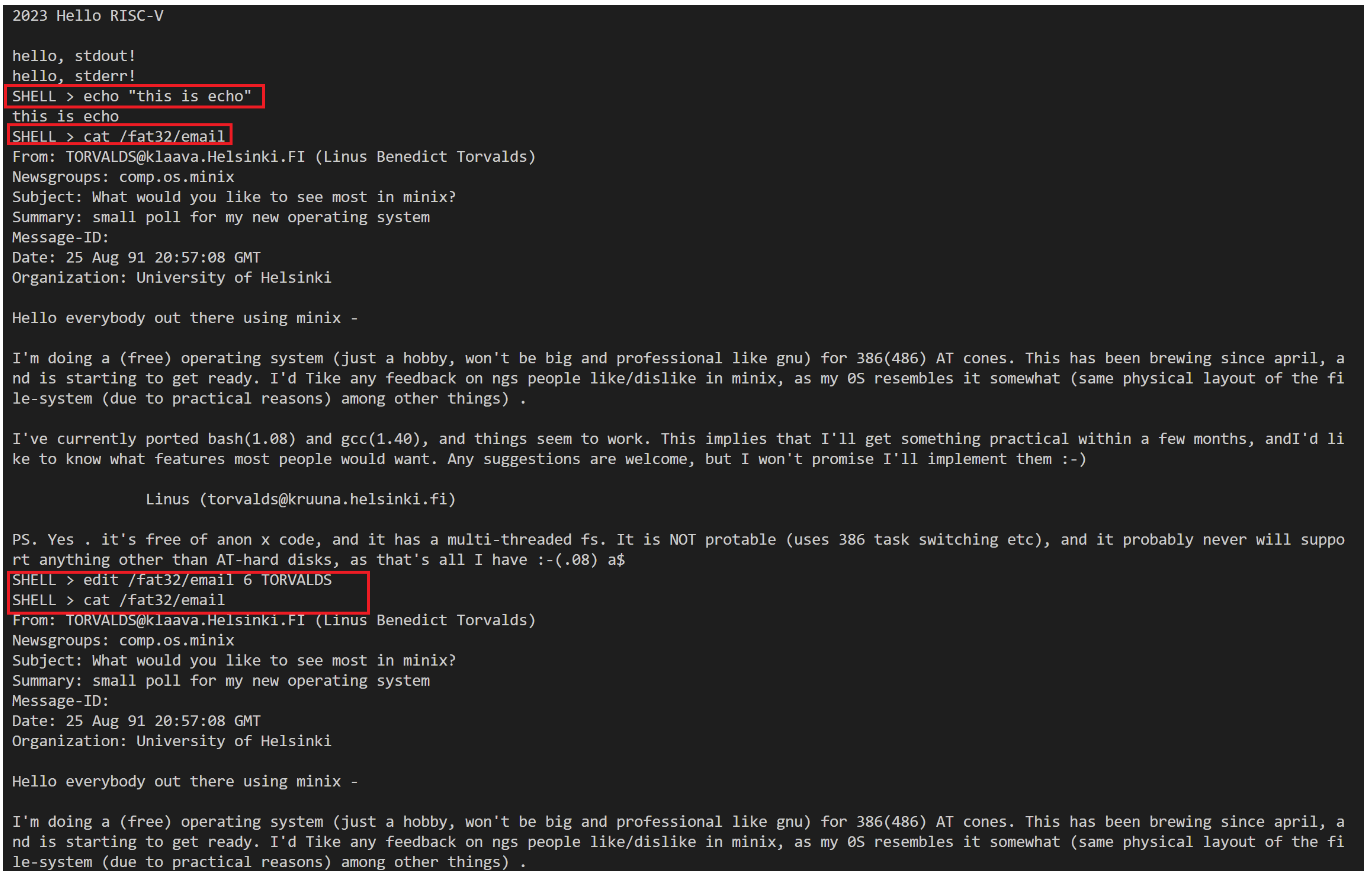
fat32_write()的实现

写操作比较简单，主要是要记得重新写回io。

```
int64_t fat32_write(struct file* file, const void* buf, uint64_t len) {
    uint64 sec = cluster_to_sector(file->fat32_file.cluster);
    virtio_blk_read_sector(sec, fat32_buf);
    // printk("%d\n", file->cfo);
    memcpy(fat32_buf+file->cfo, buf, len);
    virtio_blk_write_sector(sec, fat32_buf);
    return 0;
    /* todo: fat32_write */
}
```

实验结果

如图所示，已经将命令框选出来了，实现了交互，读文件，写入文件的操作。写后再读可以发现 torvalds改成了TORVALDS.



```
2023 Hello RISC-V

hello, stdout!
hello, stderr!
SHELL > echo "this is echo"
this is echo
SHELL > cat /fat32/email
From: TORVALDS@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID:
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT cones. This has been brewing since april, a
nd is starting to get ready. I'd like any feedback on ngs people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the fi
le-system (due to practical reasons) among other things) .

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, andI'd li
ke to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

               Linus (torvalds@kruuna.helsinki.fi)

PS. Yes . it's free of anon x code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will suppo
rt anything other than AT-hard disks, as that's all I have :-(.08) a$
SHELL > edit /fat32/email 6 TORVALDS
SHELL > cat /fat32/email
From: TORVALDS@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID:
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT cones. This has been brewing since april, a
nd is starting to get ready. I'd like any feedback on ngs people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the fi
le-system (due to practical reasons) among other things) .
```

致谢

至此，操作系统所有内容结束，话不多说，这门课让我把学过的几乎所有知识串联起来了，可以算的上一个初级professor了，可以说是醍醐灌顶，感谢老师和助教的细心讲解，感谢！