Lab5实验报告

思考题

Thinking 5.1

- 引发问题：在系统崩溃或断电的的情况下，缓存中的数据可能还未写入设备或者写回的时候终止写回，可能导致数据丢失
- 差异：对于串口设备来说，读写更加频繁，在相同时间内发生错误的概率远高于IDE磁盘

Thinking 5.2

```
git@22375080:~/22375080 (lab5)$ grep -nr "FILE_STRUCT_SIZE" ./
./fs/serv.c:394:        user_assert(sizeof(struct File) == FILE_STRUCT_SIZE);
./tools/fsformat.c:315: static_assert(sizeof(struct File) == FILE_STRUCT_SIZE);
./user/include/fs.h:24:#define FILE_STRUCT_SIZE 256
./user/include/fs.h:34: char f_pad[FILE_STRUCT_SIZE - MAXNAMELEN - (3 + NDIRECT) * 4 - sizeof(void *)];
```

- 由于一个文件控制块大小为 256B，因此一个磁盘块中最多可以存储 4KB / 256B = 16 个文件控制块
- 由于一个文件控制块最多可以直接或间接指向 1024 个磁盘块（不使用间接指针的前十个），所以最多可以有 1024 * 16 = 16384 个文件
- 1024 * 4096B = 4MB

Thinking 5.3

- 根据缓存区大小可知最大为 `DISKMAX = 0x40000000 = 1GB`

Thinking 5.4

- `fs/serv.h` 中，用于描述缓冲区以及磁盘的大小

```
/* server's address space at DISKMAP+(n*BLOCK_SIZE). */
#define DISKMAP 0x10000000

/* Maximum disk size we can handle (1GB) */
#define DISKMAX 0x40000000
```

- `user/include/fs.h` 中，有块大小的定义；文件名长度限制；路径长度限制；文件控制块中直接指针和间接指针的限制；文件最大大小；文件控制块的统一大小

```
// Bytes per file system block - same as page size
```

```
#define BLOCK_SIZE PAGE_SIZE
#define BLOCK_SIZE_BIT (BLOCK_SIZE * 8)

// Maximum size of a filename (a single path component), including null
#define MAXNAMELEN 128

// Maximum size of a complete pathname, including null
#define MAXPATHLEN 1024

// Number of (direct) block pointers in a File descriptor
#define NDIRECT 10
#define NINDIRECT (BLOCK_SIZE / 4)

#define MAXFILESIZE (NINDIRECT * BLOCK_SIZE)

#define FILE_STRUCT_SIZE 256
```

Thinking 5.5

- 会共享文件描述符和定位指针

```
int main() {
    int r;
    int fdnum;
    char buf[512];
    int n;

    if ((r = open("/motd", O_RDWR)) < 0) {
        user_panic("open /motd: %d", r);
    }
    if ((r = open("/newmotd", O_RDWR)) < 0) {
        user_panic("open /newmotd: %d", r);
    }
    fdnum = r;
    debugf("fdnum: %d\n", fdnum);

    int id;

    if ((id = fork()) == 0) {
        if ((n = read(fdnum, buf, 511)) < 0) {
            user_panic("child read /newmotd: %d", r);
        }
        struct Fd *fdd;
```

```
        fd_lookup(r,&fdd);
        debugf("child fd->offset == %d\n",fdd->fd_offset);
    }
    else {
        if((n = read(fdnum, buf, 511)) < 0) {
            user_panic("parent read /newmotd: %d", r);
        }
        struct Fd *fdd;
        fd_lookup(r,&fdd);
        debugf("parent fd->offset == %d\n",fdd->fd_offset);
    }
    return 0;
}
```

输出结果为

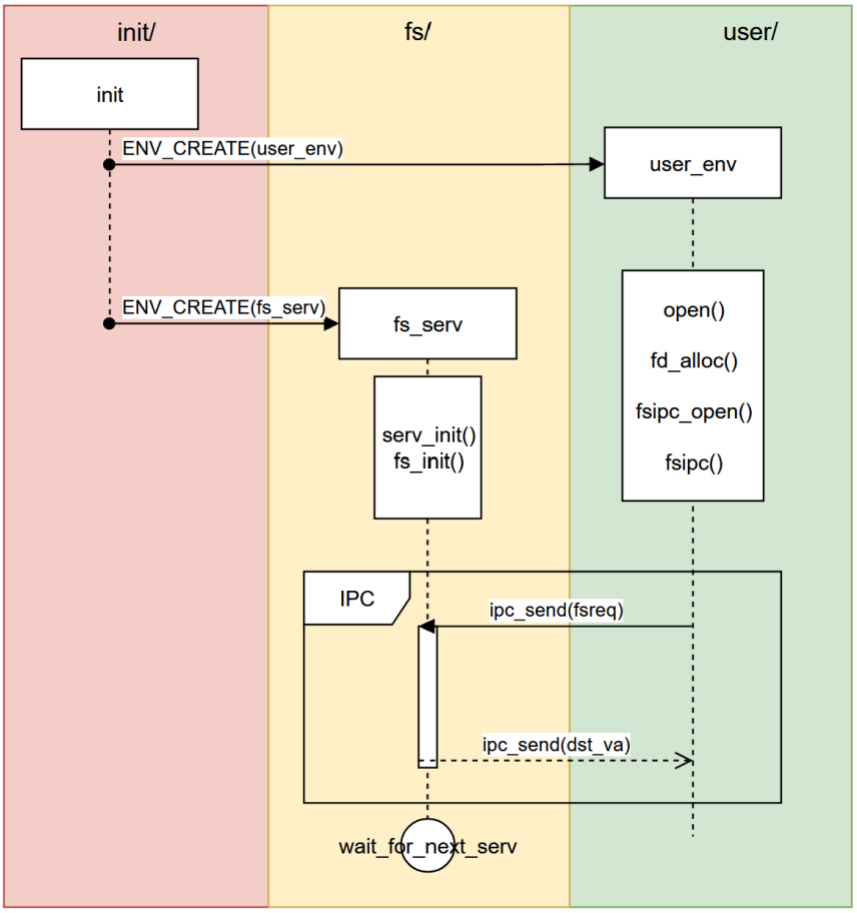
```
init.c: mips_init() is called
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
FS is running
superblock is good
read_bitmap is good
fdnum: 1
child fd->offset == 0
parent fd->offset == 0
[00001802] destroying 00001802
[00001802] free env 00001802
i am killed ...
[00000800] destroying 00000800
[00000800] free env 00000800
i am killed ...
```

Thinking 5.6

- `File` 是文件控制块，存储了文件的大部分信息，也存储了文件的磁盘块结构
- `Fd` 主要用于记录已打开的文件的状态，便于用户对文件进行操作，该结构体对应内存中的数据，描绘了磁盘块能够映射到的虚拟地址的相关信息
- `Filefd` 蕴含了更多的信息，可以借此获取文件控制块和更多文件相关信息

Thinking 5.7

- 由 `init()` 函数执行 `ENV_CREATE(user_env)` 和 `ENV_CREATE(fs_serv)`，来启动文件系统 fs 和用户进程 user
- 之后 fs 进程初始化 `serv_init()` 和 `fs_init()` 完成后，进入 `serv()` 函数，在接收信息的时候进行阻塞，等待IPC
- user 进程向 fs 进程 `ipc_send(fsreq)` 发送请求，fs 使用 `ipc_send(dst_va)` 来返回
- 之后等待下一次服务



实验难点

文件系统难点主要在于新增的代码量之大，需要好好理清楚思路和逻辑。

另外，其分为三大板块进行学习，每一个部分都有自己的出发点和实现，以及基于先前已学内容的运用

实验和课程内容结合非常紧密，但也有一个不同，需要一边学习课程内容一边进行操作系统核心的理解，并且梳理不同点

体会与感想

文件系统是区别于前边数次实验的不同的分支，并且在计组没有学过，所以算是新接触，操作系统中一些皆抽象，一些皆虚拟，真的可以给我们带来很多的震撼，操作系统中的算法和思想是之后任何系统优化中都会用到的基石！