# OSlab2 虚拟文件系统

张铭方 161220169

#### 测试说明

直接 make run 即可,按以下测试1、2、3的顺序依次完成测试

<u>测试1目的: 文件支持读写, flags支持 o rdonly , o wronly 和 o rdwr 。</u>

<u>测试2目的:在devfs中存在三个设备,null, zero, random,并且功能正常。</u>

测试3目的:文件操作API线程安全,并且有在 /proc/[pid] 下的线程统计信息

#### 功能总览

- void vfs\_init();
  - o 创建三个虚拟文件系统的实例: procfs, devfs, kvfs
- int vfs\_access(const char \*path, int mode);
  - o 判断path映射到的文件能否在mode下访问
  - 1. 查看路径头判断是哪个fs的,将剩余路径交给fs处理
  - 2. 在fs得到剩余路径后lookup找该路径映射的文件,有的话以mode判断文件inode的flag,返回正负1;没有的话返回0
- int vfs\_mount(const char \*path, filesystem\_t \*fs);
  - 。 将fs挂载到path上,并登记在path\_map\_fs中
- int vfs\_unmount(const char \*path);
  - o 将path上的fs卸载
- int vfs\_open(const char \*path, int flags);
  - 。 以flag模式打开path映射到的文件, 返回文件描述符
  - 1. 查看路径头判断是哪个fs的,将剩余路径交给fs处理,找到最小的可用文件描述符,以后拿着文件描述符可以读写文件
  - 2. 执行 vfs\_access(): 若返回1, 根据 flags 新建一个file\_t; 若返回0, 创建新的 inode, 并根据 flags 新建一个file\_t; 若返回-1, open失败。
- ssize\_t vfs\_read(int fd, void \*buf, size\_t nbyte);
  - o 对文件描述符fd对应的打开的文件,从对应的inode读取nbyte字节到buf中
  - 1. 根据fd看file t是否可读,不可则返回,可读则找到对应inode
  - 2. 从inode的file\_data的offset开始读取nbyte字节
- ssize\_t vfs\_write(int fd, void \*buf, size\_t nbyte);
  - 。 对文件描述符fd对应的打开的文件,从cur\_offset开始写nbyte字节的buf
    - 1. 看fd是否可写,不可则返回,可写则找到对应inode
  - 2. 从inode的filedata的offset开始写入nbyte字节
- off\_t vfs\_lseek(int fd, off\_t offset, int whence);

- 。 对文件描述符fd对应的打开的文件,修改cur\_offset:
  - if(whence==SEEK SET) cur offset = offset
  - if(whence==SEEK CUR) cur offset = cur offset + offset
  - if(whence==SEEK\_END) cur\_offset = end\_of\_file + offset
- int vfs\_close(int fd);
  - o 将文件描述符fd对应的打开的文件关闭

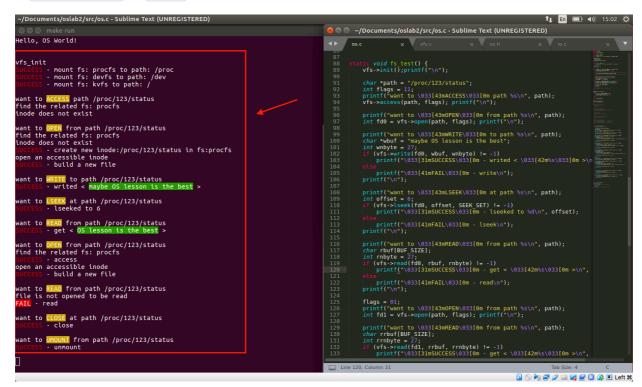
## 测试1: 可读可写文件的 创建与读写

测试目的: 文件支持读写, flags支持 o\_RDONLY, o\_WRONLY 和 o\_RDWR。

- 1. 通过 vfs->access() 检查是否存在文件及其读写权限,这里我们新建一个文件,故得到结果 inode does not exist
- 2. 通过 vfs->open() 在找到的对应 fs\_t 创建一个新的文件 inode\_t, 并创建新的打开文件file\_t, 返回文件描述符 fd0
- 3. 通过 vfs->write() 向 fd0 对应的文件里写入"maybe OS lesson is the best"
- 4. 通过 vfs->lseek() 将 fd0 对应的文件的偏移量修改为 6
- 5. 通过 vfs->lseek() 将 fd0 对应的文件从偏移量开始读取 nbyte 个字符,故得到 "OS lesson is the best"

接下来再以只写形式打开该文件、获得新的文件描述符 fd1

再 vfs->open() 就 FAIL 了, 因为没有权限。



### 测试2: 三个设备的实现与使用

测试目的: 在devfs中存在三个设备, null, zero, random, 并且功能正常

/dev/null

读取得到'\0',写入会成功

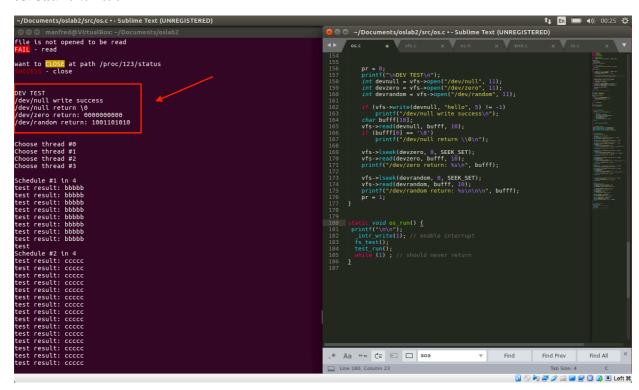
/dev/zero

读取得到无穷多个 0x00

/dev/random

读取得到无穷多个随机数

得到相应实验结果:



# 测试3: 并发进行读写操作

测试目的: 文件操作API线程安全,并且有在 /proc/[pid] 下的线程统计信息

通过信号量实现互斥锁

方法: 在 vfs->init() 中执行 kmt->sem\_init(&only, "only", 1), 之后

```
void lock() { P(&only); }
void unlock() { V(&only); }
```

#### 通过测试代码:

```
static void g(void *arg) {
   pr = 0;
   int fd = vfs->open("/kv/testfile",11);

while (1) {
   char *text = (char*)arg;
   vfs->write(fd, text, zstrlen(text));
```

```
8
            vfs->lseek(fd, 0, SEEK_SET);
 9
            char buf[BUF_SIZE];
10
            vfs->read(fd, buf, 5);
            printf("test result: %s\n", buf);
11
12
        pr = 1;
13
14
15
16
    static void test_run() {
      kmt->create(new thread(), g, (void*)"aaaaa");
17
     kmt->create(new_thread(), g, (void*)"bbbbb");
18
      kmt->create(new_thread(), g, (void*)"ccccc");
19
      kmt->create(new_thread(), see_status, (void*)NULL);
20
21
```

成功达到以下效果,说明我用信号量部分解决这个线程安全问题。

并在 see thread status 中,读取了 /proc/[pid] 下的线程统计信息。

