

# 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

**测试1目的：**文件支持读写，**flags**支持 `O_RDONLY` , `O_WRONLY` 和 `O_RDWR` 。

**测试2目的：**在devfs中存在三个设备，`null`, `zero`, `random`，并且功能正常。

**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();`
  - 创建三个虚拟文件系统的实例：procfs, devfs, kvfs
- `int vfs_access(const char *path, int mode);`
  - 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);`
  - 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);`
  - 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);
  - 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。

The left screenshot shows the output of a test program. It displays the results of various file system operations. A red box highlights the output of the `write` operation, which successfully wrote the string "maybe OS lesson is the best" to the file. A red arrow points from this output to the corresponding `write` operation in the C code on the right.

The right screenshot shows the C code implementing the file system. The code defines a `fs_test` function that performs the following operations:

- Initialize the file system with `vfs->init()`.
- Check if the file `/proc/123/status` exists using `vfs->access(path, flags)`.
- Open the file `/proc/123/status` for writing using `vfs->open(path, flags)`, returning file descriptor `fd0`.
- Write the string "maybe OS lesson is the best" to the file using `vfs->write(fd0, wbuf, wnbyte)`.
- Seek to the offset 6 using `vfs->lseek(fd0, offset, SEEK_SET)`.
- Read the string "OS lesson is the best" from the file using `vfs->read(fd0, rbuf, rnbyte)`.
- Open the file `/proc/123/status` for reading using `vfs->open(path, flags)`, returning file descriptor `fd1`.
- Close the file `fd1` using `vfs->close(fd1)`.
- Unmount the file system using `vfs->unmount()`.

## 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：

[illegible]

### 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

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

通过测试代码：

```
1 static void g(void *arg) {
2     pr = 0;
3     int fd = vfs->open("/kv/testfile", 11);
4
5     while (1) {
6         char *text = (char*)arg;
7         vfs->write(fd, text, zstrlen(text));
```

