

总结

1. link/unlink: 本质上是操作根目录下的DirEntry。如果进行link, 则查找原文件的id, 然后用新名字和id新建一个DirEntry加到根目录的disk_inode下; unlink的时候就是找到对应名字的DirEntry, 将其内容写成空的即可, 当然如果发现这是最后一个(只有一个对应id的项了), 则还需要用clear方法把文件的内容删掉。
2. get_state: 需要获取的信息通过遍历即可拿到, 操作上比较容易; 而对于指针地址转换, 方法参照task_info的写法即可。注意其中的inode_id, 需要通过block_id和block_offset反推算得到。

ch6 问答题

1. 在我们的easy-fs中, root inode起着什么作用? 如果root inode中的内容损坏了, 会发生什么?

在本次实验中, root inode是根目录的inode, 作用是查找根目录的内容, 由于我们只有一层目录, 所以root inode相当于被用于查找(根目录下)所有文件的索引。

如果root inode损坏, 则可能出现找不到其他文件inode的问题, 进而出现文件丢失的问题。

ch7 问答题

1. 举出使用pipe的一个实际应用的例子

```
ls -a ./ | grep ^a
```

上面的命令可以列出当前目录下以a开头的文件。

2. 如果需要在多个进程间互相通信, 则需要为每一对进程建立一个管道, 非常繁琐, 请设计一个更易用的多进程通信机制。

可能的办法: fork一个专门处理通信的进程(比方说从root进程fork), 其余所有进程和该进程通过管道建立通信(或者, 这个进程干脆就是所有进程的父进程。这里只是作为示意), 并附带目的进程信息, 然后这个中心进程再进行转发。这个过程最好给用户封装一下, 方便使用。这样对于n个子进程, 只需要建立n个pipe即可。

另一种办法: 内核提供一个类似send/recv的系统调用, send参数为接受方pid和发送的buffer, recv参数为发送方pid和接受buffer。

当发送方进程调用send, 内核拿到对应的buffer和接受方pid, 然后为recv进程分配一块用作缓冲区的内存(这部分内存接受方不需要能访问), 将发送方buffer的内容拷贝上去; 当接受方进程调用recv时, 如果传入的pid参数和之前发送方的pid一致(接收方想要接受来自这个发送方的消息), 则将缓冲区的内容拷贝到接受方buffer中。

这样对于用户态程序, 如果想进行进程间通信, 只需知道双方的pid并且调用send和recv即可。