

OS-lab7 report

刘泓尊 2018011446 计84

Chapter7 实现的内容

1. 阅读并理解了 `easy-fs`、`easy-fs-fuse` 的设计思路
2. 在内核中使用 `easy-fs`，自底向上构建块设备驱动层、内核索引节点层 `OSInode`、文件描述符层，并对接到系统调用层
3. 实现了简单的文件系统和目录抽象，只有一个根目录的扁平化文件系统
4. 增加了 `sys_openat`、`sys_dup`、`sys_linkat`、`sys_unlinkat`、`sys_fstat` 系统调用
5. `sys_exec` 适配了新的应用加载逻辑，直接从文件系统读取代码，而不是通过加载器获得。支持命令行参数。
6. 在用户态增加了 `cat` 命令行工具，利用 `fork+dup+exec` 来支持 `shell` 的IO重定向。

编程作业：

1. open syscall

```
pub fn sys_openat(_dirfd: usize, path: *const u8, flags: u32, _mode: u32) -> isize
```

用于打开一个文件，并返回它的文件描述符。

首先调用 `translated_str` 把 `path` 从虚拟地址区间转换成内核可读的字符串。

之后调用 `fs::open_file` 打开这个文件，并向当前进程的文件描述符表中增加该描述符。

如果 `open_file` 失败返回-1。 `open_file` 内部提供了针对flag的检查和分支判断。

2. linkat syscall

```
pub fn sys_linkat(_olddirfd: i32, oldpath: *const u8, _newdirfd: i32, newpath: *const u8, _flags: u32) -> isize
```

创建一个文件的一个硬链接，硬链接指的是多个文件名指向同一个inode，在 `easy-fs` 的实现中，指的是多个 `name` 对应同一个 `inode_id`。这可以通过向根节点的 `DiskInode` 中插入**目录项** `DirEntry` 来实现。为了维护文件的硬链接 `nlink` 数量，可以将 `nlink` 像目录项 `DirEntry` 一样序列化到磁盘，也可以简单一些，在查询 `nlink` 的时候，先得到待查询文件的 `inode id`，然后遍历根目录下所有 `inode`，统计和该 `inode id` 相同的个数，这就是这个文件的硬链接数量。

首先要把 `oldpath` 和 `newpath` 都翻译成物理地址，同时检测该地址是可读的，并解析成字符串。查找到 `oldpath` 对应的 `inode_id`。

之后创建一个新文件，这个文件不需要调用 `alloc_inode` 来新分配 `inode id`，而是直接把 `inode_id` 作为它的 `inode id`。之后创建这个目录项，插入到根目录的内容中。

这样我们就创建了一个硬链接，核心在于 **inode id 相同**，不新分配该文件的空间，而是**向根目录的磁盘块中写入新的目录项**，这样也保证了所有硬链接地位相同。

需要注意的是，创建目录项的操作需要渗透到 `DiskInode` 这一层，也就是要**实际修改磁盘上的记录**，不然OS关闭之后，所有的操作就丢失了。

3. unlinkat syscall

```
pub fn sys_unlinkat(_dirfd: i32, path: *const u8, _flags: u32) -> isize
```

取消一个文件路径到文件的链接。

首先要将 `path` 翻译成物理地址，保证地址可读，并解析成字符串 `name`。

在根目录的目录项中查找 `name` 对应的目录项，调用 `DiskInode::write_at` 将该目录项清空即可。为了实现简单，不实现缩容操作。实际上仅仅把 `name` 清空就可以让后续对 `name` 的查找失败了。

不存在就返回-1

4. fstat syscall

```
pub fn sys_fstat(fd: usize, st: *mut Stat) -> isize
```

用于获取文件状态。

为了支持对任何文件描述符都能获得索引编号 `inode_id` 和硬链接数量 `nlink`，我为 `File` Trait 增加了2个接口，这样就方便fstat系统调用的实现。

```
1 pub trait File : Send + Sync {
2     fn inode_id(&self) -> usize;
3     fn nlink(&self) -> usize;
4 }
```

首先判断 `st` 指向的内存是否可写，可写就翻译成物理地址。

之后读取当前进程描述符表中第 `fd` 个，并拿到该文件的索引编号 `inode_id` 和硬链接数量 `nlink`。硬链接数量 `nlink` 通过遍历根目录所有与待查询文件 `inode_id` 相同的 `inode_id` 目录项的个数获得。

如果文件描述符表中不存在第 `fd` 个文件，返回-1

测试结果截图

```
Ustests: Running ch7_file0
Test file0 OK!
Ustests: Test ch7_file0 in Process 1 exited with code 0
Ustests: Running ch7_file1
Test fstat OK!
Ustests: Test ch7_file1 in Process 1 exited with code 0
Ustests: Running ch7_file2
Test link OK!
Ustests: Test ch7_file2 in Process 1 exited with code 0
ch7 Ustests passed!
```

问答作业

1. 目前的文件系统只有单级目录，假设想要支持多级文件目录，请描述你设想的实现方式，描述合理即可。

`easy-fs` 库已经提供了足够的机制来实现多级目录，需要对 `Inode` 进行扩展。

一个节点是目录还是文件在 `DiskInode` 中维护，而内存中的 `OSInode` 维护的是磁盘中存放该 `DiskInode` 的索引。

与在目录中创建文件的方法 `create` 类似，可以实现一个在目录中创建目录的方法 `create_dir`，给该目录分配一个 `Inode`，并初始化为目录，`Inode` 的初始化过程如下：

```

1  // create a new file
2  // alloc a inode with an indirect block
3  let new_inode_id = fs.alloc_inode();
4  // initialize inode
5  let (new_inode_block_id, new_inode_block_offset) =
    fs.get_disk_inode_pos(new_inode_id);
6  get_block_cache(
7      new_inode_block_id as usize,
8      Arc::clone(&self.block_device)
9  ).lock().modify(new_inode_block_offset, |new_inode: &mut DiskInode| {
10      new_inode.initialize(DiskInodeType::Directory);
11  });

```

之后创建目录项 `DirEntry` 并插入当前目录。

在调用查找文件的方法 `find` 的时候，不再是只有根目录可以调用了，任何目录类型的 `Inode` 都可以调用。根据 `name` 在当前 `Inode` 对应的 `DiskInode` 中查找出对应的 `inode` 编号，创建新的 `Inode` 返回给调用者。这部分逻辑不需要改动，即 `find` 方法只支持单级查找。

为了支持多级查找，可以新增加一个接口 `find_multi`，以 `name` 参数中的 `\` 为分隔，多次调用 `find` 来实现多级查找的能力。`find_multi` 既可以被根节点调用，也可以被其他目录节点调用，从而可以实现相对路径的查询。

还有一个要解决的问题就是进入上级目录。在 `easy-fs` 的实现中，`DirEntry` 只是一个孩子指针，我们还需要给 `DiskInode` 维护的磁盘区域增加一块目录项，表示父指针 `..`，信息包括父目录的 `name` 和 `inode id`。在创建新目录时，需要在该目录磁盘块的目录项里创建指向父目录的目录项；在查找过程中遇到 `..` 便可以读取该目录项，得到父目录对应的 `inode id`，来进入父目录。

由此，我们便实现了多级目录的创建和查找。

本质上就是要把带有父指针和孩子指针的多叉树序列化到磁盘，序列化的内容包括父指针、孩子指针（目录项）、文件内容，并支持树节点的创建、查找、删除过程。

2. 在有了多级目录之后，我们就也可以为一个目录增加硬链接了。在这种情况下，文件树中是否可能出现环路？你认为应该如何解决？请在你喜欢的系统上实现一个环路，描述你的实现方式以及系统提示、实际测试结果。

允许目录的硬链接可能造成环。比如文件树 `~/dir1/dir2/`，在 `dir2/` 中创建了一个 `dir1/` 的硬链接，那么 `dir1/` 既可以看做父目录也可以看做子目录。这在用户一步步 `cd` 的时候没什么问题，但是对于 `ls`，`tree` 这类遍历文件树的软件，就会出现无限循环。所以一般OS不允许对目录创建硬链接。

除了不允许建立目录硬链接，另一种简单但成本高的解决办法就是，在遍历的过程中记录 `Inode id`，如果遍历时出现了重复就不再遍历。

在 `Ubuntu 20.04` 系统上测试硬链接环路：

```

liuhz@lh:~/temp$ tree
.
├── dir1
│   ├── 1.txt
│   └── dir2
│       └── 2.txt
2 directories, 2 files
liuhz@lh:~/temp$ cd dir1
liuhz@lh:~/temp/dir1$ cd dir2
liuhz@lh:~/temp/dir1/dir2$ ls
2.txt
liuhz@lh:~/temp/dir1/dir2$ ls ..
1.txt  dir2
liuhz@lh:~/temp/dir1/dir2$ ln .. dir1_hard
ln: ../: hard link not allowed for directory

```

可以看到 Ubuntu 20.04 不允许对目录创建硬链接。

在 win10 上测试硬链接环路（同样的目录结构）：

```

D:\大三下\temp\dir1\dir2>mklink /H dir1_hard ..
拒绝访问。

```

结果是**拒绝访问**。查看帮助其实可以发现 mklink 的硬链接 /H 也是只针对文件的，不能对目录创建。

```

D:\大三下\temp>mklink /?
创建符号链接。

MKLINK [[/D] | [/H] | [/J]] Link Target

/D      创建目录符号链接。默认为文件
        符号链接。
/H      创建硬链接而非符号链接。
/J      创建目录联接。
Link    指定新的符号链接名称。
Target  指定新链接引用的路径
        (相对或绝对)。

```

我们可以针对目录创建一个软链接 /J，看一下这时的情况：

```

D:\大三下\temp\dir1\dir2>mklink /J dir1_soft ..
为 dir1_soft <==> .. 创建的联接

D:\大三下\temp\dir1\dir2>ls
驱动器 D 中的卷是 新加卷
卷的序列号是 02E6-5F7F

D:\大三下\temp\dir1\dir2 的目录

2021/04/05  18:33    <DIR>          .
2021/04/05  18:33    <DIR>          ..
2021/04/05  18:19                5 2.txt
2021/04/05  18:33    <JUNCTION>    dir1_soft [D:\大三下\temp\dir1]
                1 个文件          5 字节
                3 个目录 304,629,141,504 可用字节

```

可以看到软链接是可以创建成功的。但是这时执行 tree 命令：

[illegible]

tree 命令出现了无限循环，所幸这个软件限制了最大深度。

所以，对目录创建链接会破坏文件树的DAG结构，OS最好应该禁止这种情况的发生。