用户态文件系统大作业报告

李晨昊 2017011466

2020 年 6 月 2 日

目录

1 简介

1.1 运行环境

我实现的是一个网络学堂映射的文件系统，后续称之为LearnFS。本项目采用Rust，基于fuse-rs库实现，在Linux平台上测试过，具体平台为：

```
$ uname -a
```

```
Linux mashplant 5.0.0-32-generic #34~18.04.2-Ubuntu SMP Thu Oct 10 10:36:02
    UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
$ rustc -V
rustc 1.44.0-nightly (f509b26a7 2020-03-18)
```

需要安装libfuse-dev和pkg-config才能正常编译。

1.2 编译及运行方式

执行cargo run -- <mountpoint>即可在<mountpoint>处挂载LearnFS，切换到这个目录即可使用相应功能，具体功能在后面介绍。debug模式下编译出的二进制文件相当大，改用release模式编译就会小很多，不过性能其实没太大差别，因为主要时间开销在网络 IO 上。

我在程序中一些地方输出了 log，可以通过export RUST_LOG=<level>来控制 log 的输出，<level>为info时输出量比较合适。

2 功能及实现思路

2.1 网络学堂接口

网络学堂部分的接口是我自己编写的，封装在一个独立的crate中，借鉴了 Harry Chen 的thu-learn-lib。相比他的接口而言，我既有做一定的简化，也有添加更多的接口。

我提供了阻塞和异步两套接口，我使用的是异步接口，即Rust中的Future那一套。文件系统接口本身是同步的，所以我的文件系统需要自带一个可以执行Future的运行时，我使用的是tokio库的运行时。

虽然文件系统的各种操作本身是同步的，但是使用异步的接口仍然可以提升性能。第一是因为异步接口的内部实现也是异步的，在内部请求一组 URL 时可以同时开始；第二是因为我可以在文件系统中一次性执行一组异步操作，例如我可以同时获取一门课的作业，通知，文件和讨论列表：

```
let (hs, ns, fs, ds) = unwrap!(self.runtime.block_on(try_join4(
  client.homework_list(&course), client.notification_list(&course),
  client.file_list(&course), client.discussion_list(&course))), reply);
```

它们的执行可以相互交织，这样效率会更高。根据测试同时获取这四个列表的耗时大约与其中耗时最长的一个相同 (一般是作业列表)。

2.2 学号登录

在LearnFS挂载的目录下执行mkdir，创建一个以学号为名称的文件夹时，会要求用户输入密码，并用这个学号和密码登录网络学堂，之后cd到这个目录，即可访问这个学生的学堂信息。目录的第一层是学期列表，cd进去即可查看对应学期的课程。课程目录下包括表示这门课的作业，通知，文件和讨论的文件夹，cd进去即可查看全部对应内容，每个内容都有自己的子内容，如表示一个作业的目录下包括作业的描述等内容：

```
/$ mkdir 2017011466
请输入密码：...
...
/2017011466/2019-2020-春/存储技术基础/作业/Key-Value存储实验$ ls -U
提交作业  刷新  描述  发布时间  截止时间  提交时间  成绩  批阅时间  批阅老师
    附件：KV存储.zip  提交附件：engine.zip
```

2.3 下载文件

表示一个作业，通知，文件的目录下都可能有对应的文件可以下载，作业目录下包括作业附件，提交附件和评语附件；通知目录下包括通知附件；文件目录下包括对应的文件。在打开相应文件时 (即执行open函数时)，会将对应文件下载下来，所以它们可以像普通文件一样直接读取。

2.4 提交作业

作业目录下有一个名为提交作业的文件，向其写入 (即执行write函数时) 时可以提交作业。如果写入的格式是"FILE=<filename> ..."，还可以提交文件名为<filename>的文件作为作业附件：

```
/2017011466/2019-2020-春/存储技术基础/作业/Key-Value存储实验$ ls -U
提交作业  刷新  描述  发布时间  截止时间  提交时间  成绩  批阅时间  批阅老师
    附件：KV存储.zip  提交附件：engine.zip
/2017011466/2019-2020-春/存储技术基础/作业/Key-Value存储实验$ echo 'FILE=发
  布时间 foo' >> 提交作业 # 提交内容为'foo'，附件为'发布时间'
```

2.5 参与讨论

讨论目录下有课程的所有讨论，每个讨论都表示为一个文件夹，cd进去即可查看全部的回复。每个回复的作者和时间信息表示在文件名中，回复内容在文件内容中，读取这个文件

即可查看回复。向一个文件写入，会发送一条对它的回复，写入的格式与提交作业的格式相同，同样支持附加。删除一个文件会尝试删除一条回复，一般而言，只有作者是自己的情况下删除才会成功。

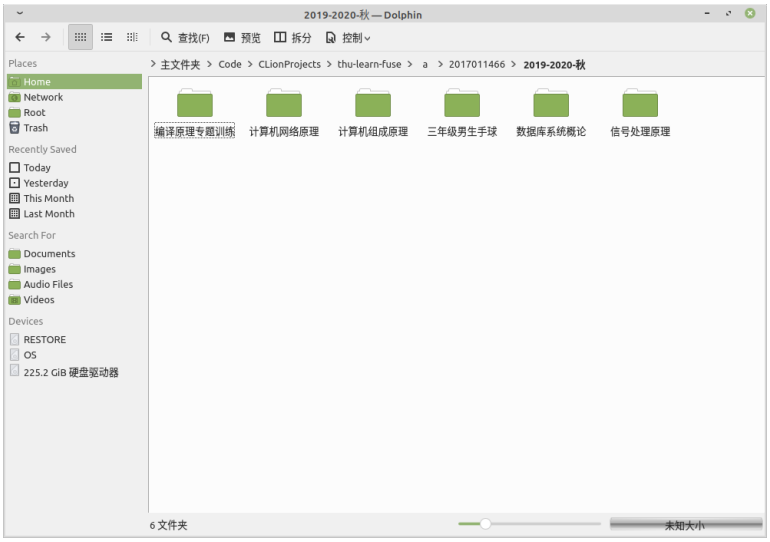
受限于当前的实现，提交作业和参与讨论的数据都不会随着修改操作立即更新，而是另外提供了一个名称为刷新的文件，向其写入任意内容则会更新本文件夹下的数据。

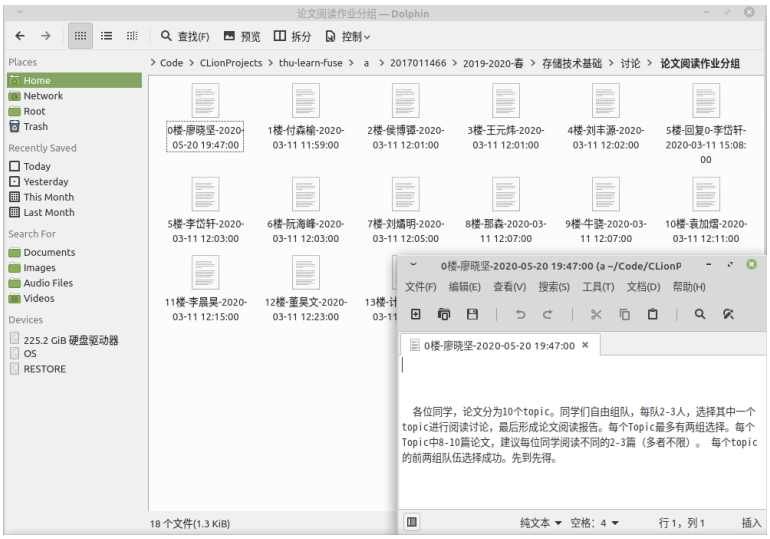
```
/2017011466/2019-2020-讨论/论文阅读作业$ ls -U
刷新
'0楼-***-2020-03-11 11:59:00'
'1楼-***-2020-03-11 11:59:00'
...
'5楼-***-2020-03-11 12:03:00'
'5楼-回复0-***-2020-03-11 15:08:00'
...
'11楼-李晨昊-2020-03-11 12:15:00'
...
/2017011466/2019-2020-讨论/论文阅读作业$ echo 'foo' >> '11楼-李晨昊
   -2020-03-11 12:15:00' # 回复我自己，内容为'foo'，会变成11楼-回复0
/2017011466/2019-2020-讨论/论文阅读作业$ echo 'foo' >> '0楼-廖晓坚
   -2020-03-11 11:59:00' # 对0楼的回复等价于添加新的楼层
```

2.6 用文件浏览器和编辑器操作

支持了基本的文件系统 API 之后，很自然地就可以用文件浏览器和编辑器来操作挂载LearnFS的文件夹。相比于之前用echo >>来写入文件，经过实验得知，文本编辑器写入的时候除了write，还会调用create(其实用用echo >来写入文件也会调用create)，因此也需要实现，实现的逻辑是只能对上面描述的那几个允许写的文件进行create，且create本身是空操作。

一些运行结果的截图如下：





3 问题及解决方式

3.1 与 libfuse 的 API 不一致

我使用的`fuse-rs`库是有对`libfuse`的 API 的封装，但主要是自己用`Rust`实现的`fuse`接口 (这是完全可以理解的，系统编程方面`C`能做到的`Rust`也能做到)。所以二者 API 有不一致是很正常的，而且`fuse-rs`已经长久没有维护了，文档也比较缺失，所以我在自己探索它的 API 的细节的时候花了一些时间。下面举几个我碰到的困难的例子：

1. `libfuse`对文件的定位都是通过路径实现的，而`fuse-rs`都是通过一个 64 位无符号整数实现的。其实从最后的实现来看，后者对实现友好得多，因为不需要自己实现路径解析的逻辑。但是我在刚接触的时候对此不是很理解，容易与表示一个文件 **handle** 编号的整数搞混，后者表示的是`open`函数的返回结果，而前者逻辑上相当于`open`函数的参数 (虽然是整数而非字符串)。

2. 一部分数据结构的定义不一样。例如libfuse中直接使用了Linux的struct stat，这在平时编程中是比较熟悉的，而fuse-rs的对应结构是struct FileAttr，与之有一定区别，例如加入了一些 (据文档说是)macOS中独有的字段，需要根据它的文档和示例来填写。

3. readdir的readdir函数可以忽略offset，简单地返回所有的子文件，fuse_fill_dir_t的offset参数传 0 即可。但是在fuse-rs中似乎没有办法绕过，文档里也没有写明到底可不可以，所以返回子文件的时候必须考虑offset的影响。

3.2 输入密码的实现

为了输入密码，不能直接在LearnFS中从标准输入读取密码，因为这样不能从请求的发起进程输入密码，不符合人的使用直觉。所以需要在请求的发起进程的标准输出中输出输入密码的提示信息，以及从它的标准输入中读取密码。fuse-rs提供了获取请求的发起进程的进程号 (pid) 的接口，然后可以利用procfs，/proc/<pid>/fd/0就是请求的发起进程的标准输出，/proc/<pid>/fd/1就是标准输入。直接打开对应路径，就可以像操作普通文件一样操作其他进程的输出和输入：

```
fn get_password(pid: u32) -> std::io::Result<String> {
  ...
  let mut stdin = BufReader::new(
    File::open(format!("/proc/{}/fd/0", pid))?);
  let mut password = String::new();
  stdin.read_line(&mut password)?;
  Ok(password.trim().to_owned())
}
```

3.3 提交文件的实现

提交文件本身没有什么复杂之处，理论上只需要读取对应的文件，再调用提交的接口就可以了。问题出在读取文件上。

第一个问题是，在LearnFS中执行文件系统操作时，当前目录 (cwd) 总是LearnFS的程序的当前目录，而不是请求的发起进程的当前目录，这样也不符合人的使用直觉。解决方法同样是使用procfs，/proc/<pid>/cwd是一个指向进程<pid>的当前目录符号链接，所以只需要打开这个路径，再以这个路径为基准执行读取文件即可。

第二个问题是，因为fuse-rs是单线程的 (libfuse默认也是)，所以当一个请求尚未结束时，后续的请求无法开始，所以如果在自定义的文件系统操作中再访问文件系统，那么就

有可能导致死锁。这一点在fuse的文档中也有提到：

> Note that single-threaded mode also means that you will not have to worry about reentrancy, though you will have to worry about recursive lookups. In single-threaded mode, FUSE holds a global lock on your filesystem, and will wait for one callback to return before calling another. This can lead to deadlocks, if your script makes any attempt to access files or directories in the filesystem it is providing.

而如果要使整个文件系统支持多线程访问 (另一个Rust库fuse-mt支持)，则在所有地方都需要考虑线程安全带来的影响，所以这里就不做这个尝试了。我采用的解决方法是，先结束当前的请求，而在一个另外的线程中执行需要访问文件系统的操作：

```
self.runtime.spawn(async move {
  ...
  let dir = Dir::open(format!("/proc/{}/cwd", pid))?;
  let mut file = dir.open_file(path)?;
  ...
});
```

这样需要考虑线程安全的地方就很少，而且因为我在其他地方也使用了tokio的运行时，所以这里并不需要执行派生线程这样昂贵的操作，可以使用现有的线程池。

这样处理的一个弊端在于，结束请求时仍然无法知道后续的操作能否成功，所以即使发生错误，也无法返回错误码。目前我没有想到什么好的解决方案，只能自己在 log 里记录一下。