lab4

第六章引入了文件系统代替了原先全部加载入内存的形式。

整个文件系统抽象层很细致 层间接口都进行了讲解 但是由于抽象层级比较多 所以理解起来比较困难 综合来看 这一章是我认为难度最大的章节

首先是对前面章节实现的系统调用的支持 这里除了因为文件系统变化而使得读取ELF数据发生变化的spawn系统调用 其余系统调用几乎不怎么需要修改 故略去不谈

这里重点来说一下我在使用文件系统实现spawn时候产生的Bug

在Lab3中 我直接利用TCB的new方法根据读取内存获得的elf_data创建新的进程 并且**我直接在syscall里面设置父子关系 这一点在Lab3中是可行的 切换到Lab4后 我直接修改了读内存的方法为读文件 但是Bug就这样水灵灵的发生了 在读取进行几次后 读取文件的接口会直接读到0字节 非常突然**

我一度以为是文件系统中出了问题 直到我和朋友交流 他使用的是类似的做法 但是他把父子关系设置这一部分放在了Task模块里面 把他和syscall解耦了 我突然意识到 是不是因为我过强的耦合 导致一些所有权/借用问题在这里面发生了 我尝试了解耦 完成后直接就Work了 但我没有想明白这里是哪个地方出现了问题 时间原因我先记录下来 等到完成全部Lab后回头尝试解决这个问题

那么完成向前兼容后，就开始来做文件系统的三个syscall

这里的主要难点是要读懂文件系统的代码，理清楚每个抽象层负责的内容，这样才能知道在哪里进行修改、添加逻辑

我的修改都发生在VFS层，并且向OS提供接口和支持，使得OS能够获得链接相关的信息

在这部分里面 我思考/查阅了一些对link信息的维护方法 对于大型系统/大量信息 我认为应该采用专用数据结构来维护link信息 尽管我对linkat的期待是从OS层看上去没有区别 但是由于unlinkat和fstat的要求OS必须得知该inode是否是linked的 所以需要额外信息来进行维护 但这里我偷了个懒 直接把inode_id设置为无意义值来让fstat在检查的时候得知其已经失效 但显然 一个最大的问题是 这样没法在应该删除文件的时候把他删除 But dont care that 因为这份代码里的底层抽象层并没有向上提供删除接口 所以这里我也暂时不考虑

这个Lab虽然最难 但带给我的收获也很大

1. 从系统思维的角度来说 这个Lab对我的抽象能力进行了锻炼 让我对抽象特别是系统中的抽象实现和抽象意义有了更深层的理解
2. 从系统实现的角度来说 这个Lab让我对文件系统有了基本的了解 知道文件系统的工作方式与基本结构 初步理解文件系统
3. 从Rust学习的角度来说 这个Lab中文件系统的实现使用了大量的闭包以及一些线程安全的包裹 使我对Rust语言以及闭包的理解加深