JOS 设计文档（lab5范围内）
丁卓成 5120379064

一、概述

本文档为Lab1-4提交的设计文档之续，继续分析JOS的架构，本次Lab的主要内容是实现文件系统。

二、文件系统的实现

本次Lab中的文件系统设计地较为简化，省去了inode层，直接将文件的metadata置于目录文件中，不支持hard link或soft link。就文件系统对硬盘的组织而言，由IDE驱动(fs/ide.c)，buffer cache（fs/bc.c）以及文件操作（fs/fs.c）组成。这里值得注意的是，它将整个硬盘映射到了虚拟地址空间，并利用page fault handler进行懒惰加载，称之为buffer cache（尽管没有有效的驱逐机制）。

由于JOS采用exokernel的设计思想，文件系统作为一个进程存在，所以在fs/fs.c的文件操作上搭建了一层server层（fs/serv.c），处理其他用户进程的请求。从用户操作的fd到最终文件的映射关系和Unix系统较为类似：每个进程拥有若干打开的fd，一个fd对应着某种设备上打开的某个文件，不同的fd可以对应相同的打开文件（通过fork或dup2）；每个打开文件对应着一个实际的文件或连接，以及offset等相关信息，不同的打开文件可以对应相同的实际文件。对于前一个层次的映射关系，我们为每个打开文件赋予一个物理页，存放一个struct Fd，不同的fd对应到不同（或相同）进程中的虚拟页，最后可以映射到同一个物理页。对于后一个层次的映射关系，由fs server维护一张映射表opentab，将struct Fd同对应的struct File联系起来。在现代的类Unix操作系统中，上述第三层为v-node，即virtual inode，底下可以由不同的文件系统支撑，而在本Lab中显然只有一种文件系统因此不需要额外的层次。

上述内容Lab已经实现提供了大部分代码，为完成练习只需参照其余部分填写少数未完成的功能即可。

三、加载存放于文件系统的程序

Lab中默认提供的方式为spawn调用，由父进程为子进程从文件系统中读取ELF段并映射到其地址空间，由于加载者和被加载者的分离，可以容易地在用户态实现。为了完成练习，只需实现sys_env_set_trapframe调用即可将其实现补充完整。此外，作为Challenge，我还实现了用户态的exec调用。由于exec调用中加载者与被加载者相同，通常需要借助内核来进行加载工作，为了避免内核的参与，我将所需用到的函数和数据结构单独存放到了.lib和.libdata两个section中，两者一起作为一个段.lib加载到虚拟地址0x08000000（具体实现参见answers-lab5.txt）。这样一来，尽管限制了所用到的这些函数和数据结构不可做成模块化可替换的设计，但确实起到了一种hypervisor的作用，使得用户态的exec可以实现。我参照icode添加了user程序icode2，其区别仅是将spawn调用换成了exec调用，经测试和icode输出相同，证明实现基本正确。

四、结论

此前在CSE课程上介绍过文件系统的实现，也做过yfs的Lab，又在OS课上继续强调，加之本Lab本身代码量设计得就不大，因此本Lab相对而言可说是最简单的了，我前后只花了一天时间就将其完成。本Lab中体现的新内容不多，主要是复习巩固性质的，故以上仅做了简要分析，更多内容待Lab6再行探讨。